INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Équipe Ostre

Optimisation des Systèmes Temps Réel Embarqués

Rocquencourt

THÈME 1C

Table des matières

1. Composition de l'équipe

L'action de recherche OSTRE est issue de la partie du projet SOSSO en charge des travaux sur la méthodologie adéquation algorithme-architecture pour le contrôle-commande et le traitement du signal en temps réel.

Responsable scientifique
Yves Sorel [DR Inria]

Assistante de projet
Martine Verneuille [SAR Inria]

Ingénieurs associé et experts
Christophe Macabiau [ingénieur associé Inria, jusqu'au 30/08/2002]
Frédéric Gager [ingénieur expert Inria, à partir du 1/11/2002]
Julien Forget [ingénieur expert Inria, à partir du 1/12/2002]

Chercheurs doctorants
Liliana Cucu [boursière Inria]
Nicolas Pernet [boursier Inria]
Hamoudi Kalla [boursier Inria, co-encadrement BIP/OSTRE]
Linda Kaouane [co-encadrement ESIEE/OSTRE]
Mickaël Raulet [co-encadrement INSA/MITSUBISHI ELECTRIC ITE/OSTRE]

Stagiaires
Nicolas Dos Santos [IUT Vélizy]
Julien Forget [DEA/DESS Université Bordeaux 1]
Frédéric Gager [Ingénieur ESIEE Noisy-Le-Grand]
Nicolas Pernet [DEA Université Paris 6]

Collaborateurs extérieurs
Rémy Kocik [Professeur assistant ESIEE]
Thierry Grandpierre [Professeur assistant ESIEE]
Christophe Lavarenne [Ingénieur UBIC]

2. Présentation et objectifs généraux

Nous menons des travaux sur l'optimisation des systèmes distribués temps réel embarqués selon quatre axes de recherche :

1. la modélisation de tels systèmes à l'aide de la théorie des graphes et des ordres partiels,
2. l'optimisation d'implantations à l'aide :
   - d'algorithmes d'ordonnancement temps réel dans le cas monoprocesseur,
   - d'heuristiques de distribution et d'ordonnancement temps réel dans le cas multiprocesseur,
   - d'heuristiques de minimisation de ressources dans le cas circuit intégré et multiprocesseur,
3. les techniques de génération automatique de code pour processeur (exécutif dédié ou configuration d'OS résident) et circuit intégrés spécifiques (« net-list »),
4. la tolérance aux fautes.

Bien que depuis le début notre thématique ait été nourrie à l'INRIA par les méthodes et applications pour l'automatique étudiées dans le projet SOSSO, nous avons toujours activement participé aux travaux effectués autour des langages synchrones. Les nouvelles applications auxquelles nous nous intéressons, principalement

dans le milieu des transports et des télécommunications, nous amènent à étendre nos axes de recherche vers la tolérance aux fautes et la conception conjointe (« codesign ») avec partitionnement logiciel/matériel automatique. Nos modèles ont été étendus ces dernières années pour prendre en compte d'une part des architectures multiprocesseur hétérogènes et/ou des circuits intégrés spécifiques conduisant à la notion de multicomposant en vue de la conception conjointe logiciel/matériel. D'autre part nous cherchons à prendre en compte, sur les fonctionnalités à implanter, des contraintes de dépendance, et des contraintes temps réel multiples de cadences (période des capteurs) et de latences (temps de réponse entre entrée-capteur et sortie-actionneur ou temps d'exécution d'un ensemble de fonctions dépendantes). Les contraintes temps réel multiples nous ont conduit à intensifier nos recherches dans le domaine de l'ordonnancement où les problèmes que nous cherchons à résoudre sont assez peu étudiés. Les techniques de génération d'exécutif maintenant bien maîtrisées sont en train d'être étendues à la génération automatique de code VHDL pour la synthèse de circuits intégrés spécifiques.

Ces travaux de recherche ont conduit d'une part à une méthodologie appelée AAA pour « adéquation algorithme-architecture » et d'autre part à un logiciel de CAO niveau système pour le prototypage rapide et l'implantation optimisés de systèmes distribués temps réel embarqués, appelé SynDEx. Ils évoluent pour permettre la tolérance aux fautes et la conception conjointe avec partitionnement logiciel/matériel automatique.

Tous ces travaux sont réalisés avec l'objectif de faire le lien entre l'automatique et l'informatique principalement dans le cadre des langages synchrones en cherchant à supprimer la rupture entre la phase de spécification/simulation et celle d'implantation temps réel, ceci afin de réduire le cycle de développement des applications distribuées temps réel embarquées.

3. Fondements scientifiques

3.1. Introduction

Mots clés : *contrôle*, *commande*, *traitement du signal*, *traitement d'image*, *prototypage rapide*, *codesign*, *CAO système*, *langages synchrones*, *multiprocesseur*, *parallèle*, *distribution*, *répartition*, *placement*, *temps réel*, *embarqué*, *optimisation*, *ordonnancement*, *exécutif*, *circuit intégré spécifique*, *circuit intégré reconfigurable*.

Participants : Liliana Cucu, Thierry Grandpierre, Rémy Kocik, Christophe Lavarenne, Yves Sorel.

La méthodologie AAA vise le prototypage rapide et l'implantation optimisés d'applications distribuées temps réel embarquées devant être tolérantes aux fautes, telles celles rencontrées en contrôle-commande de systèmes complexes comprenant du traitement du signal et des images. Elle est fondée sur des modèles de graphes, autant pour spécifier les Algorithmes applicatifs et les Architectures matérielles distribuées comportant un certain niveau de redondance, que pour déduire les implantations possibles en termes de transformations de graphes. L'Adéquation revient à résoudre un problème d'optimisation consistant à choisir une implantation dont les performances, déduites des caractéristiques des composants matériels, respectent les contraintes temps réel et d'embarquabilité, et exploitent la redondance matérielle pour effectuer de la tolérance aux fautes par redondance logicielle automatique. Dans le cas du temps réel critique les approches « hors ligne » sont privilégiées, et quand des approches « en ligne » sont utilisées, le nombre de décisions prises lors de l'exécution est minimisé, c'est-à-dire uniquement quand elles sont inévitables. Tout cela permet de générer automatiquement d'une part des exécutifs distribués temps réel à faible surcoût pour les composants processeurs, et d'autre part des « net-list » pour les composants circuits intégrés spécifiques, supportant tous ensemble (« codesign ») l'exécution tolérante aux fautes de l'algorithme sur l'architecture. La méthodologie AAA est concrétisée par un logiciel de CAO niveau système nommé SynDEx pour le prototypage rapide et l'implantation optimisés de systèmes distribués temps réel embarqués. Afin de réduire la rupture entre la phase de spécification/simulation des automaticiens et la phase d'implantation en temps réel des informaticiens, et afin de minimiser la durée du cycle de développement des applications distribuées temps réel embarquées, les liens entre des langages haut niveau orientés métier et AAA/SynDEx sont étudiés.

3.2. Contexte et objectifs

Nos recherches concernent la mise en œuvre efficace de systèmes électroniques numériques programmables (logiciel) ou non programmables (circuit intégrés spécifiques) pour des applications de contrôle-commande et de traitement du signal et des images, soumises à des contraintes temps réel et d'embarquabilité devant être tolérantes aux fautes, comme on en rencontre dans les domaines du transport (avionique, automobile), des télécommunications, etc...[7]

Dans ces applications, le système programmé et/ou câblé effectue le « contrôle-commande » [1] de son environnement en produisant, par l'intermédiaire d'actionneurs (réactions), une commande qu'il calcule à partir de son état interne et de l'état de l'environnement, acquis par l'intermédiaire de capteurs (stimuli). C'est en ce sens que l'on parle de *systèmes réactifs* [16] : la commande est calculée en réaction à chaque stimulus. Une analyse mathématique du système de commande et de son environnement permet d'une part de réaliser une spécification fonctionnelle conduisant à l'algorithme (ou l'ensemble d'algorithmes) qu'il faudra implanter, et d'autre part de réaliser une spécification de contraintes. En ce qui concerne les contraintes temps réel, on détermine une borne supérieure sur le délai qui s'écoule entre deux échantillons sur chaque capteur (cadence ou période), et une borne supérieure sur la durée du calcul (latence) entre une détection de variation d'état de l'environnement et la variation induite de la commande. Si ces bornes ne sont pas respectées l'environnement risque de ne plus pouvoir être contrôlé, ceci pouvant conduire à des conséquences catastrophiques. Ainsi, sauf indication contraire, quand nous parlons par la suite de temps réel nous faisons allusion au « temps réel strict » ou « temps réel dur » par opposition au « temps réel souple » où il est seulement question d'exécuter au mieux les applications sans réellement respecter une contrainte temporelle. En plus de ces contraintes temps réel, l'application est soumise à des contraintes technologiques d'embarquabilité et de coût, qui incitent à minimiser les ressources matérielles (architecture) nécessaires à sa réalisation. Pour satisfaire les contraintes temps réel, et/ou pour rapprocher les ressources de calcul le plus près possible des capteurs et des actionneurs afin de limiter le coût du câblage et l'influence des radiations électromagnétiques, l'architecture est la plupart du temps *multicomposant* (parallèle, répartie, distribuée), composée de plusieurs processeurs et circuits intégrés spécifiques (ASIC[2] figé ou FPGA[3] reconfigurable).

La complexité des applications visées, au niveau des algorithmes, de l'architecture matérielle, et des interactions avec l'environnement sous contraintes temps réel, nécessite des méthodes pour minimiser la durée du cycle de développement, depuis la conception jusqu'à la mise au point des prototypes ainsi que des « produits de série » obtenus à partir de ces prototypes, s'exécutant dans les deux cas en temps réel. Afin d'éviter toute rupture entre les différentes phases du cycle de développement et pour permettre des vérifications formelles et des optimisations, notre méthodologie AAA de *prototypage rapide et d'implantation optimisés* est fondée sur une approche globale, formalisant l'algorithme, l'architecture et les implantations possibles, à l'aide de graphes. L'intérêt principal de ce modèle réside dans sa capacité à spécifier à la fois du *parallélisme potentiel* dans le cas de l'algorithme (par exemple schémas-blocs utilisés dans Simulink) et du *parallélisme effectif* dans le cas de l'architecture (par exemple réseaux de composants programmables ou de fonctions logiques utilisés dans VHDL structurel), mais aussi de décrire l'implantation de l'algorithme sur l'architecture en termes de transformations de graphes (distribution et ordonnancement des calculs et des communications, synthèse des chemins de données et de contrôle), en supportant naturellement la notion de conception conjointe logiciel/matériel ou « codesign ». Ceci permet d'assurer un haut niveau de « sécurité de conception » grâce à la cohérence des modèles, de poser plus précisément les problèmes d'optimisation de l'implantation, et enfin de simplifier et rendre plus efficace la génération automatique de code (exécutifs distribués temps réel et/ou « net-list »).

[1] Néologisme indiquant qu'une machine à états finie enchaîne des lois de commandes, par la suite on utilisera pour simplifier le verbe contrôler pour dire que l'on effectue du contrôle-commande

[2] ASIC : Application Specific Integrated Circuit

[3] FPGA : Field Programmable Gate Array

3.3. Modèle d'algorithme

La spécification fonctionnelle d'une application conduit à un algorithme obtenu en général par composition de plusieur algorithmes. Par exemple dans une application de robotique mobile un algorithme de commande de trajectoire peut utiliser dans sa boucle de rétroaction un algorithme de traitement d'image effectuant de la vision active et des algorithmes de commande pour les moteurs électriques. Un algorithme, tel que défini par Turing et Post, est une séquence (ordre total) finie d'opérations directement exécutable par une machine à états finie. Cette définition doit être étendue afin de permettre d'une part la prise en compte du parallélisme effectif des architectures distribuées, composées de plusieurs machines à états finies interconnectées, et d'autre part la prise en compte de l'interaction infiniment répétitive de l'application avec son environnement (systèmes réactifs). Pour cela notre modèle d'algorithme est un *graphe de dépendances de données conditionné factorisé* [9] : c'est un hypergraphe orienté acyclique (DAG) [15], dont les sommets sont des *opérations* partiellement ordonnées [25] (parallélisme potentiel) par leurs dépendances de données (hyperarcs orientés pouvant avoir plusieurs extrémités pour une seule origine, « diffusion de données »). Chaque opération du graphe peut être à son tour décrite par un sous-graphe permettant une spécification hiérarchique de l'algorithme jusqu'aux « opérations atomiques » que l'on ne peut spécifier à l'aide d'un sous-graphe. Des *dépendances de conditionnement* permettent de choisir, à chaque réaction (répétition infinie), parmi un ensemble de sous-graphe alternatifs, celui qui sera exécuté. Le choix s'effectue suivant la valeur de la dépendance de conditionnement. L'imbrication des conditionnements conduit à de la hiérarchie dans le graphe d'algorithme. À chaque interaction avec l'environnement, concrétisée par un ensemble d'événements d'entrée issus des capteurs, les valeurs des arcs de conditionnement déterminent l'ensemble des opérations à exécuter pour obtenir les événements de sortie pour les actionneurs, à partir des valeurs d'entrée acquises par les capteurs. Chaque capteur produit une suite d'événements et chaque actionneur consomme une suite d'événements. La notion d'événement est prise au sens large, en effet la séquence infinie d'événements formant un *signal* peut être périodique ou apériodique. Chaque événement a une valeur prise dans un ensemble bien défini (les entiers ou les réels par exemple). L'algorithme est donc modélisé par un graphe de dépendances de données infini, mais pour lequel on peut identifier un motif infiniment répété. Cela permet de réduire le graphe par factorisation à son motif répété qui est généralement appelé *graphe flot de données* [19]. Les événements valués « circulent » sur les arcs de ce graphe qui sont des signaux formant les flots de données. Chaque sommet non conditionné produit ses événements sur ses signaux de sortie dès que tous les événements correspondant à ses signaux d'entrées s ont arrivés. De plus, un sous-graphe du graphe d'algorithme peut être répété un nombre fini de fois et peut contenir, à son tour, un sous-graphe lui aussi répété un nombre fini de fois correspondant à des « nids de boucles ». L'imbrication des boucles conduit aussi à de la hiérarchie dans le graphe d'algorithme. Un sous-graphe répété un nombre fini de fois peut aussi être réduit par factorisation à son motif répétitif. Les frontières de factorisation sont matérialisées par des sommets particuliers qui servent à spécifier si la répétition est infinie ou finie, et dans ce dernier cas quel est le nombre de répétitions, ainsi qu'à spécifier le début ou la fin d'une factorisation. Dans le cas d'un sous-graphe factorisé représentant un sous-graphe répété N fois, le sommet de factorisation *Fork*, qui a comme entrée un tableau de N données, va produire sur ses N sorties les N éléments de ce tableau. Chacun des éléments va être utilisé comme entrée par chacune des répétition finie de sous-graphes. Inversement, le sommet *Join* qui a comme entrées les N éléments obtenus en sorties des N instances de répétition, produit en sortie le tableau formé de ces N éléments. Un Fork et un Join appartenant à une même frontière de factorisation correspondent au déroulement spatial d'une boucle (itération ou répétition temporelle) conduisant à du « parallélisme potentiel de données » puisque c'est le même graphe d'opérations répété spatialement qui traite des données différentes. Ce type de parallélisme est un cas particulier du « parallélisme potentiel d'opérations » ou « parallélisme potentiel de tâches », induit par l'absence de dépendance de donnée entre opérations, qui est le seul parallélisme potentiel évoqué jusqu'à présent. On peut faire le même raisonnement pour les frontières de répétitions infinies. Dans ce cas, pour chaque signal d'entrée il faudra disposer d'une infinité de capteurs placés en parallèle, de même pour chaque signal de sortie il faudra disposer d'une infinité d'actionneurs, plutôt que d'un seul capteur et un seul actionneur, réutilisés infiniment. On comprend aisément que l'on peut transformer une répétition spatiale

en répétition temporelle, en transformant la répétition spatiale en une itération, ce qui diminue les ressources nécessaires, et inversement on peut transformer une itération en une répétition spatiale si les ressources le permettent.

Le conditionnement et la factorisation sont les équivalents, en termes de graphe de dépendances de données, des structures de contrôle respectivement « If...Then...Else » et « For i=1 to N Do... » que l'on trouve dans les langages impératifs. Ils ont comme principal avantage de permettre d'exprimer du parallélisme potentiel.

Le graphe de l'algorithme peut être soit directement spécifié comme tel, ou bien déduit d'une spécification séquentielle ou CSP (Communicating Sequential Processes de Hoare) par analyse de dépendances, ou encore produit par les compilateurs des langages synchrones (Esterel, Lustre, Signal, à travers leur « format commun DC ») [21] qui présentent l'intérêt de faire des vérifications formelles en termes d'ordre sur les événements. De manière plus générale, tout langage de « haut niveau orienté métier » possédant une sémantique compatible avec celle des langages synchrones, est un candidat capable de produire lors de sa compilation un graphe de ce type. Par exemple, le langage AIL (Architecture Implementation Language) défini par les constructeurs et équipementiers français du domaine de l'automobile, le langage Scicos orienté vers la spécification d'applications d'automatique, le langage AVS orienté vers la spécification d'applications de traitement d'images, sont des langages de ce type.

3.4. Modèle d'architecture

Nous distinguons deux types d'architecture, celles que nous appelons *multicomposant* qui correspondent à l'interconnexion de composants programmables (processeurs) et de composants non programmables (circuits intégrés spécifiques), et celles des circuits intégrés spécifiques eux-mêmes.

Les modèles les plus classiquement utilisés pour spécifier une architecture multiprocesseur (multicomposant ne comportant pas de circuit intégré spécifique) parallèle ou distribuée, sont les PRAM (Parallel Random Access Machines) et les DRAM (Distributed Random Access Machines) [26]. Le premier modèle correspond à un ensemble de processeurs communicant par mémoire partagée alors que le second correspond à un ensemble de processeurs à mémoire distribuée communicant par passage de messages. Si ces modèles sont suffisants pour décrire, sur une architecture homogène, la distribution et l'ordonnancement des opérations de calcul de l'algorithme, ils ne permettent pas de prendre en compte des architectures hétérogènes ni de décrire précisément la distribution et l'ordonnancement des opérations de communications inter-processeurs qui sont souvent critiques pour les performances temps réel.

Notre modèle d'*architecture multicomposant* hétérogène [4] est donc un graphe orienté, dont chaque sommet est une machine à états finie (machine séquentielle) [20] et chaque arc une connexion physique entre deux machines à états finies. Il y a cinq types de sommets : l'*opérateur* pour séquencer des opérations de calcul (séquenceur d'instructions), le *communicateur* pour séquencer des *opérations de communication* (canal DMA), le *bus/mux/démux* avec ou sans *arbitre* pour sélectionner, diffuser et éventuellement arbitrer des données, la *mémoire* pour stocker des données et des programmes. La mémoire peut aussi être considérée comme une machine séquentielle. Il y a deux types de sommets mémoire : la mémoire RAM (à accès aléatoire) pour stocker les données ou programmes locaux à un opérateur, la RAM et la SAM (à accès séquentiel), toutes deux pour les données communiquées entre opérateurs ou/et communicateurs. L'arbitre, quand il y en a un dans un bus/mux/démux/arbitre, est aussi une machine à états finie qui décide de l'accès aux ressources partagées que sont les mémoires. Les différents sommets ne peuvent pas être connectés entre eux de n'importe quelle manière, il est nécessaire de respecter un ensemble de règles. Par exemple deux opérateurs ne peuvent pas être connectés directement, de même pour deux communicateurs. Ils peuvent chacun être connecté à une RAM partagée ou à une SAM pour communiquer, en passant, ou non, par l'intermédiaire de communicateurs pour assurer le découplage entre calcul et communication. L'hétérogénéité ne signifie pas seulement que les sommets peuvent avoir chacun des caractéristiques différentes (par exemple durée d'exécution des opérations et taille mémoire des données communiquées), mais aussi que certaines opérations ne peuvent être exécutées que par certains opérateurs, ce qui permet de décrire aussi bien des composants programmables (processeurs) que des composants non programmables (ASIC ou FPGA). Un processeur est décrit par un sous-graphe

contenant un seul opérateur, une ou plusieurs RAM de données et de programme locaux. Un moyen de communication direct (sans routage) entre deux processeurs, est un sous-graphe contenant au moins une RAM (données communiquées) et des bus/mux/démux/arbitre, ou bien un sous-graphe linéaire composé au minimum des sommets (bus/mux/démux/arbitre, RAM, communicateur, RAM ou SAM, communicateur, RAM, bus/mux/démux/arbitre).

Notre modèle d'architecture de circuit intégré spécifique est le modèle classique RTL (Register Transfer Level, niveau transfert de registres) [24]. C'est un graphe orienté ou chaque sommet est un circuit combinatoire ou une mémoire, et chaque arc est un transfert de données entre circuit et mémoire.

Afin d'unifier le modèle multicomposant et le modèle de circuit intégré nous avons étendu le modèle RTL que nous qualifions alors de *Macro-RTL*. Une opération du graphe de l'algorithme correspond alors à une *macro-instruction* (une séquence d'instructions pouvant se réduire à une seule instruction dans le cas d'un composant non programmable, ou bien un circuit combinatoire) ; une dépendance de donnée correspond à un *macro-registre* (des cellules mémoire contiguës ou des conducteurs interconnectant des circuits combinatoires). Ce modèle encapsule les détails liés au jeu d'instructions, aux micro-programmes, au pipe-line, au cache, et lisse ainsi ces caractéristiques de l'architecture, qui seraient sans cela trop 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.

3.5. Modèle d'implantation

L'*implantation* d'un algorithme sur une architecture multicomposant est une *distribution* et un *ordonnancement* non seulement des opérations de l'algorithme sur les opérateurs de l'architecture, mais aussi des opérations de communication, qui découlent de la première distribution, sur les communicateurs, les bus/mux/démux/arbitre et les mémoires.

La distribution consiste à affecter chaque opération de l'algorithme à un opérateur capable de l'exécuter. Ceci conduit à une partition de l'ensemble des opérations de l'algorithme en autant de sous-graphes qu'il y a d'opérateurs. Ensuite pour chacune de ces opérations il faut ajouter un sommet *alloc* d'allocation mémoire programme locale (resp. allocation mémoire données locales) et affecter ce sommet à une RAM programme (resp. RAM données locales) connectée à l'opérateur qui exécute l'opération. Enfin, il faut affecter chaque dépendance de donnée inter-opérateur (c'est-à-dire entre opérations affectées à des opérateurs différents), à une *route* reliant les deux opérateurs (chemin dans le graphe de l'architecture), créer et insérer, entre les deux opérations de l'algorithme, autant d'opérations de communication qu'il y a de communicateurs, autant de sommets *identité* qu'il y a de sommets bus/mux/démux/arbitre et autant de sommets *alloc* d'allocation de mémoires données communiquées, qu'il y a de sommets mémoire SAM et RAM sur la route, puis les affecter aux sommets correspondant du graphe de l'architecture. Ceci conduit à une partition de l'ensemble des sommets de communication, des sommets *identité* et des sommets *alloc* respectivement en autant de sous-graphes qu'il y a de commmunicateurs, de bus/mux/demux et de mémoires. Les sommets *alloc* permettent de déterminer la taille des mémoires nécessaires pour l'application.

L'ordonnancement consiste à linéariser (rendre total par ajout d'arcs) l'ordre partiel associé à chaque sous-graphe de l'algorithme formé d'opérations, d'opérations de communication, de sommets *identité* et *alloc*, affecté à un sommet respectivement opérateur, communicateur, bus/mux/demux, et mémoires du graphe de l'architecture, car ceux-ci sont des machines séquentielles.

Une implantation est donc le résultat d'une transformation du graphe de l'algorithme (ajout de nouveaux sommets et de nouveaux arcs) en fonction du graphe de l'architecture, lui même transformé (détermination de toutes les routes possibles) [4]. L'ensemble de toutes les implantations possibles, étant donnés un algorithme et une architecture, est formalisé comme une composition de trois relations binaires : le *routage*, la *distribution* et l'*ordonnancement*, chacune d'elles mettant en correspondance deux couples de graphes (algorithme, architecture) [12]. On peut aussi la voir comme une loi de composition externe où un graphe d'algorithme est composé avec (influencé par) un graphe d'architecture pour donner comme résultat un graphe d'algorithme transformé (distribué et ordonnancé). Chacune de ces implantations possibles a des performances (latence, cadence)

différentes. Ces performances sont obtenues par calcul de chemins critiques (pour les latences) et/ou de boucles critiques (pour les cadences) sur le graphe de l'implantation étiqueté par les durées d'exécution caractéristiques des opérateurs, des communicateurs, des bus/mux/démux/arbitre et des mémoires de l'architecture.

L'*implantation* d'un algorithme sur une architecture de circuit intégré spécifique [2] est une transformation des opérations de l'algorithme en circuits combinatoires formant le *chemin de données* du circuit, et une transformation des dépendances de conditionnements et des sommets de factorisation en circuits combinatoires et mémoires réalisant des circuits séquentiels formant le *chemin de contrôle* du circuit. Les sommets de factorisation spécifient une répétition spatiale de sous-graphes. Chaque répétition spatiale peut être transformée partiellement ou totalement en répétition temporelle si nécessaire. Les dépendances de données sont transformées en transferts de données entre circuits. Contrairement à l'implantation sur un multicomposant, le graphe de l'algorithme n'est pas ici influencé par un graphe d'architecture, en effet celui-ci n'est pas spécifié. Ainsi le graphe d'algorithme est transformé directement en un graphe d'implantation qui est le graphe d'architecture du circuit visé.

3.6. Optimisations

La recherche d'une implantation optimisée d'un algorithme sur une architecture multicomposant ou de circuit intégré spécifique, tenant compte de contraintes temps-réel et d'embarquabilité, correspond à une adéquation (« mise en correspondance efficace ») entre cet algorithme et cette architecture, celle-ci pouvant être le résultat de l'implantation dans le cas d'un circuit intégré. Nous effectuons actuellement trois types d'optimisation : l'optimisation de la distribution et de l'ordonnancement dans le cas d'un multicomposant, l'optimisation de l'implantation dans le cas d'un circuit intégré, et l'optimisation de l'ordonnancement dans le cas monoprocesseur. Les deux premiers types d'optimisation utilisent des heuristiques, car les problèmes traités sont NP-difficiles conduisant à des solutions approchées, tandis que le troisième type utilise des algorithmes exacts conduisant à des solutions optimales. Les optimisations d'implantation se font « hors-ligne », c'est à dire avant l'exécution en temps réel de l'application. Cette approche est bien adaptée au contexte des applications temps réel qui doivent être déterministes afin d'assurer une bonne sécurité de conception, de plus elle conduit à une génération de code induisant un surcoût inférieur à celui obtenu par des optimisations faites « en-ligne », c'est à dire pendant l'exécution en temps réel de l'application.

Le problème d'optimisation de la distribution et de l'ordonnancement dans le cas multicomposant que nous avons formalisé [8][10], et dont nous avons automatisé la résolution approchée, se limite au cas de l'adéquation entre un algorithme et une architecture donnés, y compris dans leur granularité et leur topologie, sous une contrainte temps réel unique de cadence égale à la latence et sans préemption d'une opération par une autre opération. Même ainsi réduit, ce problème est reconnu NP-difficile, et le nombre d'implantations possibles dans le cas d'une application réaliste rend prohibitive toute tentative de recherche exhaustive de la solution optimale, c'est pourquoi on utilise des heuristiques pour trouver des solutions approchées. De plus, l'objectif de prototypage rapide nous a fait étudier plus particulièrement des heuristiques « gloutonnes » (sans retour arrière) qui s'exécutent très rapidement [22].

Les heuristiques de distribution-ordonnancement que nous développons sont du type « gloutonnes » et de « list-scheduling » (choix fait par les fonctions de coût à partir des opérations ordonnançables), améliorées pour prendre en compte les communications inter-opérateurs qui peuvent avoir lieu par mémoire partagée ou par passage de messages, l'hétérogénéité des opérateurs, des communicateurs, des bus/mux/démux/arbitre et des mémoires, et enfin le conditionnement des opérations. Les communications inter-opérateur sont distribuées et ordonnancées sur les routes, en tenant compte avec précision des parties de routes communes, des routes parallèles et des conflits d'accès aux ressources partagées principalement les séquenceurs des opérateurs requis par les communicateurs [6]. L'optimisation de l'utilisation des différents types de mémoire (RAM programme ou données et SAM) est réalisée par ré-allocation statique. En ce qui concerne la hiérarchie et la répétition finie de sous-graphe de l'algorithme, nous la traitons en effectuant de la défactorisation partielle sur nos graphes factorisés (transformation spatiale/temporelle). Pour cela, nous nous basons sur les travaux effectués sur la parallélisation de nids de boucles dans le cas des langages impératifs. Enfin, ces heuristiques sont ensuite

étendues à des heuristiques itératives de voisinage local [11] qui ne sont plus « gloutonnes » (avec retour arrière), beaucoup plus lentes mais donnant des résultats plus précis. Cela consiste à mémoriser les endroits où l'heuristique gloutonne se trouve en face de choix équivalents, puis à revenir sur ces choix en explorant toutes les possibilités plutôt que d'en choisir une au hasard et de continuer comme cela est fait dans les heuristiques « gloutonnes ».

La recherche d'une implantation optimisée d'un algorithme sur une architecture de circuit intégré spécifique tenant compte de contraintes temps-réel et d'embarquabilité, est aussi un problème complexe qui est en général NP-difficile. Elle consiste à appliquer des transformations telles que celles décrites plus haut en évaluant, à chaque transformation à l'aide d'une fonction de coût, si les contraintes temps réel et d'embarquabilité sont respectées. Plus concrètement la fonction de coût calcule la latence du circuit et sa surface ou le nombre de CLB (fonctions logiques) du FPGA relativement à une caractérisation en termes de durée et de surface, de nombre de fonctions logiques des éléments de bibliothèque VHDL synthétisables selon la technologie du circuit intégré visée. Les heuristiques que nous développons sont aussi « gloutonnes » afin d'obtenir des résultats approchés le plus rapidement possible. Nous développons aussi des heuristiques de type « recuit simulé » fondées sur le même type de fonction de coût beaucoup plus lentes, mais plus précises.

Afin de prendre en compte des contraintes temps réel multiples nous avons repris les travaux classiques sur l'ordonnancement monoprocesseur pour les reformuler dans le cadre de notre modèle d'algorithme de graphe conditionné factorisé, en exprimant la notion de périodicité à l'aide de la factorisation vue comme une répétition de graphe temporelle (par opposition à spatiale). Nous avons ainsi pu proposer un algorithme optimal d'ordonnancement [1] d'un système temps réel sans préemption sur monoprocesseur avec contraintes multiples de précédences de données entre une opération productrice et une ou plusieurs opérations réceptrices, de périodicités chacune appliquée à une opération, et de latences chacune appliquée à un couple d'opérations. Cet algorithme est optimal dans le sens ou s'il existe un ordonnancement l'algorithme le trouvera. Pour être plus précis, comme le graphe sur lequel l'algorithme d'ordonnancement s'applique est infini, celui-ci s'arrête dès qu'il ne peut plus ordonnancer une opération, ce qui veut dire dans ce cas que le système n'est pas ordonnançable.

La conception conjointe logiciel/matériel consite à choisir la partie de l'algorithme qui sera implantée en logiciel c'est-à-dire sur un multiprocesseur (composants programmables), et celle qui sera implantée en matériel c'est-à-dire sur un ou plusieurs circuits intégrés spécifiques (composants non programmables). Ce partitionnement en deux éléments de partition « le logiciel » et « le matériel » est généralement fait de façon manuelle. Parce que nous avons un cadre formel unique pour décrire algorithme, composants matériels programmables et composants matériels non programmable, il est plus facile de poser un problème d'optimisation pour le partitionnement, afin que ce dernier puisse s'effectuer de manière automatique et non plus manuelle. Cela consiste à étudier comment on peut unifier les fonctions de coût utilisées dans les heuristiques pour les implantation multicomposant et les heuristiques pour circuits intégrés. Afin de faire automatiquement le choix du partitionnement nous cherchons des fonctions de coût qui prennent en compte des critères de type flexibilité, consommation électrique, etc ...en plus des critères déjà utilisés de type performances temps réel (latence, cadence) et ressources (nombre de processeurs et de moyen de communications, surface d'un circuit intégré, nombre de CLB dans un FPGA).

3.7. Génération d'exécutif et de « net-list »

Un *exécutif distribué temps réel dédié* pour processeur [5], comme une « net-list » pour circuit, correspond au codage d'une implantation particulière d'un algorithme suivant le modèle macro-RTL de l'architecture. La séquence d'opérations sur chaque opérateur ou chaque communicateur d'une architecture multicomposant est codée par une séquence de macro-instructions. Sur un circuit intégré les macro-instructions sont des composants de bibliothèque VHDL, connectés en pipeline. Le générateur d'exécutifs du logiciel SynDEx transforme donc le graphe flot de données de l'implantation en graphe flot de contrôle, codé par un exécutif et/ou des « net-list » en VHDL structurel.

La génération automatique d'exécutifs se fait suivant des règles décrivant la transformation d'un graphe d'implantation optimisé en un graphe d'exécution. Pour chaque opérateur (resp. chaque communicateur) on construit un programme séquentiel formé de la séquence des opérations de calcul (resp. des opérations de communication) qu'il doit exécuter. Les opérations de communications sont des « SEND » et des « RECEIVE » de données transmises entre communicateurs via une SAM (communication par passage de message), ou des « WRITE » et des « READ » quand les données sont transmises via des RAM (communication par mémoire partagée). Pour garantir les précédences d'exécution entre les opérations appartenant à des séquences de calcul et/ou de communication différentes, et pour garantir l'accès en exclusion mutuelle aux données partagées par les opérations de ces séquences, on ajoute des opérations de synchronisation avant et après chaque opération qui lit (resp. écrit) une donnée écrite (resp. lue) par une opération appartenant à une autre séquence. Ces opérations de synchronisation utilisent des sémaphores générés automatiquement. Nous avons montré à l'aide des réseaux de Pétri que ces sémaphores permettent à l'exécutif de respecter l'ordre partiel du graphe d'algorithme initial [4], n'introduisant ainsi pas de dead-lock dans une itération infinie ou finie donnée ou entre deux itérations infinies ou finies consécutives.

Il y a autant d'exécutifs générés, chacun d'eux correspondant à un fichier distinct, qu'il y a d'opérateurs dans l'architecture. Chaque fichier d'exécutif est un code intermédiaire indépendant de l'opérateur, c'est-à-dire du processeur puiqu'il n'y a qu'un opérateur par processeur, composé d'une liste d'appels de macros qui seront traduites par un macro-processeur en autant de programmes dans le langage source préféré (C, ou assembleur par exemple). Chacun de ces programmes source sera compilé puis chargé dans la mémoire programme de l'opérateur correspondant. Les définitions de macros qui sont dépendantes de l'opérateur (du processeur) peuvent être classées en deux ensembles. Le premier ensemble est un jeu extensible de *macros applicatives* réalisant les opérations de l'algorithme. Le second ensemble, que nous appelons *noyau d'exécutif*, est un jeu fixe de *macros système* qui supportent le chargement initial des mémoires programmes, la gestion mémoire (allocation statique, copies et fenêtres glissantes de macro-registres), le séquencement (sauts conditionnels et itérations finies et infinies), les transferts de données inter-opérateurs (macro-opérations de communication transférant le contenu de macro-registres), les synchronisations inter-séquences (assurant l'alternance entre écriture et lectures de chaque macro-registre partagé entre séquence de calcul et séquences de communication), et le chronométrage (pour permettre la mesure des caractéristiques des opérations de l'algorithme et des performances de l'implantation).

3.8. Tolérance aux fautes

Dans les applications qui nous concernent si les contraintes temps réel ne sont pas respectées l'environnement pour lequel on produit une commande ne peut plus être contrôlé, cela peut avoir des conséquences catastrophiques telles que la destruction du système de commande lui-même, pouvant conduire dans certains cas jusqu'à la perte de vies humaines. L'objet de la *sécurité de fonctionnement* est d'éviter au maximum ces problèmes. Elle se décline selon deux aspects, d'une part la *sécurité de conception* du système de commande lui-même à l'aide de techniques de vérifications formelles (model-checking, theorem proving), et d'autre part la *tolérance aux fautes*. Le premier est traité d'une part par les compilateurs des langages synchrones qui produisent une spécification d'algorithme vérifiée, et d'autre part par l'implantation optimisée sans « dead-lock » qui garantit les vérifications précédentes. Nous traitons le second d'une part en demandant à l'utilisateur de spécifier les fautes matérielles qu'il souhaite tolérer par redondance au niveau de l'architecture (processeurs, capteurs, actionneurs, réseau), et d'autre part en déduisant de cette redondance matérielle la redondance d'opérations (redondance logicielle) nécessaire au niveau de l'algorithme et au niveau de l'exécutif, afin de garantir que si des fautes surviennent malgré tout lors du fonctionnement temps réel, le comportement de l'application reste correct jusqu'à ce que toute la redondance matérielle soit exploitée. Au delà de cette limite un fonctionnement dégradé, qui doit être impérativement spécifié dans l'algorithme, sera nécessaire. Concernant la tolérance aux fautes, le problème de l'identification des fautes elles-mêmes reste très difficile à traiter.

Nous cherchons à étendre les heuristiques d'optimisation de distribution et d'ordonnancement vues plus haut pour prendre en compte la tolérance aux fautes. Pour cela, pour l'instant, seules les fautes permanentes

sont prises en compte. Ces fautes sont détectées au moyen de « chien de garde » dont la durée est calculée en fonction des durées d'exécution des opération de calcul et de communication. Nous travaillons dans deux directions : la première pour laquelle nous avons des résultats théoriques solides ne prend en compte que les fautes des opérateurs (processeurs) du graphe d'architecture, les moyens de communications étant supposés fiables, la seconde, qui est beaucoup plus complexe, prend en compte les fautes des opérateurs et des moyens de communications. Cette deuxième direction est fondée sur un calcul des distributions-ordonnancements possibles relativement aux différents cas de fautes acceptés, qui sont ensuite fusionnés en supprimant toutes les solutions redondantes. Cependant, nous avons encore du travail théorique à effectuer avant de pouvoir proposer une heuristique fiable. En ce qui concerne la première direction nous proposons deux types d'heuristiques : le premier type utilise la redondance spatiale des opérations de calcul et des opérations de communication dans le cas de moyens de communications point-à-point [3], la seconde utilise la redondance spatiale des opérations de calcul et la redondance temporelle des opérations de communication dans le cas de moyens de communications multi-point.

4. Domaines d'application

Mots clés : *télécommunications*, *automobile*, *robotique mobile*, *traitement d'image*, *traitement du signal*.

Les travaux de recherche que nous menons concernent certains laboratoires académiques utilisateurs de nos résultats sur l'ordonnancement ou de nos méthodes de prototypage rapide et d'implantation optimisés permettant le « codesign » pour réaliser des applications nouvelles, particulièrement complexes, que les méthodes traditionnelles ne permettent pas d'envisager habituellement dans le cadre de la recherche. Ils concernent aussi les industriels concevant et/ou réalisant des systèmes distribués temps réel embarqués complexes. On trouve ces derniers principalement dans les domaines du transport et plus particulièrement de l'automobile dans lequel nous nous sommes particulièrement impliqués ces dernières années, de la robotique mobile (AGV Automated Guided Vehicule) et plus particulièrement du CyCab développé en collaboration avec ROBOSOFT, des télécommunications (nouvelles normes de téléphone mobile UMTS, compression d'image JPEG2000 ou de vidéo MPEG4, radio logicielle), du traitement du signal multicapteur (radar, sonar), ou du traitement d'image (navigation automatique). Dans le cadre de projets nationaux, européens, ou de contrat spécifiques nous avons des collaborations avec les principaux industriels de ces domaines. Nos travaux intéressent aussi les éditeurs de logiciels positionnés sur le créneau des environnements logiciels d'aide à la conception et à la réalisation d'applications distribuées temps réel complexes.

5. Logiciels

5.1. SynDEx

Mots clés : *CAO niveau système*, *prototypage rapide*, *implantation*, *heuristique d'optimisation*, *génération de code processeur*, *génération de VHDL.*

Participants : Julien Forget, Thierry Grandpierre, Christophe Macabiau, Yves Sorel.

Le logiciel de CAO niveau système *SynDEx* concrétise la méthodologie AAA pour le prototypage rapide et l'implantation optimisés d'applications temps réel embarquées. Il permet de spécifier l'algorithme d'application et l'architecture multicomposant, de faire une adéquation correspondant à une implantation optimisée de l'algorithme sur l'architecture, dont le résultat est une simulation temporelle de l'exécution de l'algorithme sur l'architecture. Il génère automatiquement pour chaque processeur un exécutif temps réel dédié, ou un fichier de configuration pour un exécutif temps réel résident standard. Les exécutifs dédiés sont produits à partir de bibliothèques de noyaux d'exécutif extensibles et portables dépendant des processeurs de l'architecture. Actuellement il supporte les architectures multiprocesseur à base de stations de travail UNIX, de processeurs i80x86, de processeurs de traitement du signal TMS320C40, TMS320C60 et ADSP21060, de microcontrôleurs MPC555, MC68332 et i80C196. Il permet de faire naturellement de la conception conjointe

logiciel/matériel en connectant à ces processeurs des circuits intégrés spécifiques ASIC ou FPGA contenant une interface de communication adéquate. SynDEx est utilisé aussi bien par des universitaires que par des industriels. Il est distribué gratuitement sur le web à l'url : http://syndex.org.

5.2. SynDEx-IC

Participants : Julien Forget, Thierry Grandpierre, Linda Kaouane, Yves Sorel.

Nous développons, en collaboration avec l'équipe A2SI de l'ESIEE, le logiciel de CAO de circuit intégrés *SynDEx-IC*, pour la conception et la réalisation de composants non programmables de type ASIC ou FPGA. Il permet de spécifier l'algorithme d'application sous la même forme que dans SynDEx, et de synthétiser automatiquement le chemin de donnée et le chemin de contrôle du circuit intégré, conduisant à un programme VHDL structurel synthétisable. Ce dernier est actuellement utilisé pour générer du code pour FPGA de la famille Xilinx. Des composants réalisés avec SynDEx-IC peuvent à leur tour être utilisés dans SynDEx pour spécifier des architectures multicomposants comportant des composants programmables et non programmables.

Actuellement les deux logiciels sont séparés, mais nous visons à terme l'intégration de SynDEx-IC dans SynDEx afin d'offir un environnement unique couvrant l'ensemble du cycle de développement pour le prototypage rapide et l'implantation optimisés permettant le « codesign » d'applications très complexes.

6. Résultats nouveaux

6.1. Modèles d'algorithme, d'architecture et d'implantation

Participants : Liliana Cucu, Julien Forget, Frédérique Gager, Thierry Grandpierre, Nicolas Pernet, Yves Sorel.

Afin de traiter au même niveau les répétitions finies et infinies dans les graphes d'algorithmes nous avons défini une fonction permettant de décrire toutes les combinaisons possibles d'arcs de dépendance reliant des opérations répétitives. Ce nouveau modèle permet ainsi de décrire des algorithmes très complexes consommant et produisant des données à des rythmes différents. Cela généralise dans la théorie des graphes le modèle SDF (Synchronous Data Flow) proposé dans l'environnement logiciel Ptolemy par E. Lee [17]. Nous avons ainsi pu donner un sens précis à la notion de périodicité associée à chaque opération que l'on retrouve traditionnellement dans le modèle maintenant classique proposé par Liu et Layland [23] d'un ensemble de tâches temps réel, ces dernières correspondant à notre notion d'opération (sommet du graphe d'algorithme). Nous étudions actuellement la possibilité d'introduire dans ce modèle les nouveaux arc de données sans précédence et de précédence sans données que nous avons proposés l'année dernière.

La notion d'échéance (deadline) du modèle classique n'étant pas suffisante pour exprimer des contraintes temps réel entre une entrée et une sortie, ce qui est souvent nécessaire dans les applications industrielles (par exemple temps s'écoulant entre l'appui sur la pédale de frein et l'arrêt des roues freinées qui peut entraîner l'exécution d'un ensemble d'opérations réalisant la commande appropriée), nous avons donné une définition de la contrainte de latence appliquée à un couple quelconque d'opérations reliées par un chemin orienté [18]. Les opérations d'un tel couple peuvent donc, a fortiori, des opérations d'entrée et de sortie. Nous avons établi un lien entre la notion de chemin orienté dans un graphe d'algorithme et la notion de contrainte de latence appliquée à un couple d'opérations de ce graphe. Cela nous a servi dans la recherche des conditions d'ordonnançabilités que nous avons données par la suite.

Nous avons étudiés comment les arcs de données sans précédence pouvaient être utilisés pour délimiter des sous graphes d'algorithmes n'étant pas temps réel critique car ils n'imposent plus d'ordre entre les données produites par la partie critique et consommées par la partie non critique. Nous avons utilisé ce principes pour spécifier des opérations « boîte noire » pour lesquelles on ne peut assurer un fonctionnement critique. Les « boîtes noires » ont été testées dans le cas d'opérations d'entrées associées à des capteurs à ultrason utilisés dans une application d'arrêt sur obstacle implantée sur notre CyCab.

Nous avons exploité le conditionnement, qui sert à exprimer les différentes alternatives de sous-graphes possibles relativement à un arc de conditionnement, dans le cadre de la spécification de machines à état finies servant à décrire l'enchaînement d'algorithmes de calcul. Du point de vue de la spécification, pour chaque sommet du graphe d'algorithme toutes les données d'entrées doivent être présentes pour que le sommet puisse être exécuté afin de produire ses données de sorties. Cela veut dire que l'on considère les données d'entrée d'une opération comme toujours présentes, c'est-à-dire remises à jour à chaque réaction ou répétition infinie du graphe d'algorithme, et donc les sorties de l'opération sont elles aussi remises à jour à chaque réaction. Cependant pour un conditionnement dont les différents sous-graphes alternatifs peuvent produire des sorties différentes, mais où pour l'instant toutes les sorties sont produites à chaque réaction. Afin d'avoir une sémantique d'interface cohérente avec les langages synchrones Esterel et Signal, nous étudions l'introduction de la notion d'« événement absent », ou plutôt de « non-événement » relativement à un autre signal, dans notre modèle d'algorithme afin d'exploiter cette propriété lors de l'implantation pour n'exécuter que ce qu'il est nécessaire d'exécuter. Enfin, nous avons proposé une traduction bien adaptée à l'implantation sur des architectures distribuées, dans le sens où elle minimise les communications interprocesseur, d'un graphe flot de contrôle représentant une machine à état finie en un graphe de dépendances données conditionné factorisé. Ceci conduit à un modèle unifié permettant de faire cohabiter des spécifications issues de langages de description de diagramme d'état (dit aussi de comportement) comme StateChart, StateFlow de Simulink, ou SyncChart d'Esterel Technology, etc, et des spécifications issues de langages de description de calculs scientifiques comme Matlab-Simulink, Scicos, C, etc.

Nous avons utilisé les mémoires RAM de données pour la spécification d'architectures multi-DSP à mémoire partagées. Nous sommes en train d'étudier comment faire évoluer ce modèle de mémoire afin de mieux prendre en compte les caractéristiques fines des hiérarchies de mémoire complexes (cache, mémoires internes et externes, bans multiples) des nouveaux DSP. Nous avons utilisé les mémoires SAM (Sequential Access Memory) de données pour la spécification d'architectures à mémoire distribuées utilisant du passage de message, dans le cas de bus diffusant et non diffusant.

Le modèle d'implantation dans le cas multicomposant a évolué pour prendre en compte les nouvelles fonctionnalités, des modèles d'algorithme et d'architecture, décrites ci-dessus. Il faut noter que notre modèle d'implantation est implicitement à mémoire rémanente. En effet, lors de la tranformation de graphe qui permet de passer du graphe d'algorithme influencé par le graphe d'architecture, au graphe d'implantation, nous créons autant de sommets d'allocation qu'il y a de dépendance de données, ces sommets d'allocation ayant pour rôle lors de l'ultime transformation conduisant à la génération de code, de créer une variable partagée par le code de l'opération productrice et le code de l'opération consommatrice. Ainsi cette variable garde sa valeur tant qu'elle n'est pas modifiée lors d'une écriture par l'opération productrice. Dans le cas de la synthèse de circuit intégré, nous avons étendu le modèle d'implantation précédent, qui ne prenait en compte que l'aspect répétition (nid de boucles), afin qu'il puisse aussi prendre en compte l'aspect conditionnement. Ceci permet de synthétiser un chemin de contrôle, maintenant complet, traitant à la fois la répétition et le conditionnement.

6.2. Optimisations

En ce qui concerne l'optimisation de la distribution et de l'ordonnancement dans le cas multicomposant, nous avons modifié les heuristiques pour prendre en compte les nouvelles fonctionnalités du modèle d'algorithme et les avons principalement améliorées pour qu'elles s'exécutent plus rapidement.

La plupart de nos efforts ont porté sur l'optimisation de l'ordonnancement monoprocesseur. Nous avions proposé l'année dernière un algorithme optimal d'ordonnancement d'un système temps réel sans préemption sur monoprocesseur avec contraintes multiples de précédences, de périodicités et de latences. Nous avons donné une condition d'ordonnançabilité, fondée sur une hyper-période (PPCM de l'ensemble des périodes de toutes les opérations du graphe d'algorithme), pour un système temps réel sans préemption sur monoprocesseur avec contraintes multiples de précédences, et de périodicités. Cela permet d'arrêter l'algorithme d'ordonnancement optimal dès que l'hyper-période est atteinte. Par ailleurs, nous avons donné une condition d'ordonnançabilité, fondée sur des relations entre chemins reliant des couples d'opérations sur lesquels sont

définies les latences, pour un système temps réel sans préemption sur monoprocesseur avec contraintes multiples de précédences et de latences. Nous travaillons actuellement à donner une condition d'ordonnançabilité pour un système temps réel sans préemption sur monoprocesseur avec contraintes multiples de précédences, de périodicités et de latences, en nous servant du résultat obtenu l'année dernière établissant une relation entre périodicité et latence. Nous cherchons aussi à étendre le résultat obtenu pour des périodicités strictes à des périodicités avec « gigue », ce qui est plus réaliste. Nous envisageons d'étudier les conséquences de l'introduction de la préemption dans ces ordonnancements.

6.3. Génération d'exécutif et de « net-list »

Nous avons principalement modifié la génération automatique d'exécutif dédiés pour prendre en compte les nouvelles fonctionnalités du modèle d'architecture à savoir les mémoires de données partagées et les bus diffusants. Nous avons aussi étudié, en collaboration avec le laboratoire COSI de l'ESIEE, la configuration de systèmes d'exploitation temps réel résidents et libres tels que RTlinux et RTAI.

Nous avons poursuivi les travaux en collaboration avec le laboratoire A2SI de l'ESIEE pour développer un générateur de « net-list » basé sur du VHDL structurel dans le cas des circuits reconfigurables FPGA de Xilinx. Le VHDL produit est utilisé pour synthétiser le code de configuration des FPGA à l'aide de l'outil de synthèse LeonardoSpectrum de Mentor Graphics.

6.4. Tolérance aux fautes

Les travaux en collaboration avec le projet BIP, qui font suite à notre ARC TOLERE, ont consisté à améliorer l'heuristique proposée les années précédentes car elle engendrait trop de communications interprocesseur. Cette heuristique utilise la redondance spatiale des opérations de calcul et des opérations de communication dans le cas de moyens de communications point-à-point. Elle est fondée sur la réplication automatique des opérations du graphe de l'algorithme en fonction des ressources matérielles que l'on accepte de voir tomber en panne. Elle a été implantée dans SynDEx. Nous l'avons comparée à la méthode la plus proche existant dans la littérature mais qui est restreinte, par rapport à la notre, aux architectures multiprocesseur homogènes. Pour cela nous avons réalisé un générateur aléatoire de graphes d'algorithme et d'architecture qui nous a permi d'une part de montrer l'efficacité de notre heuristique, et d'autre part de mesurer le surcoût dû à la tolérance aux fautes par rapport au cas où on ne la prend pas en compte.

6.5. Logiciel SynDEx

La version 6 du logiciel SynDEx, maintenant programmée en Caml/Tk plutôt que C^{++}, offre les nouvelles fonctionnalités de spécification d'algorithme avec hiérarchie, conditionnement et répétition. Elle a été mise au milieu de l'année en distribution gratuite sur le web, sous copyright INRIA. Elle est accompagnée d'un manuel d'utilisation, d'un tutorial, d'un jeu d'exemples représentatifs, d'une bibliothèque d'opérations de calcul scientifique inspirée de celle de Simulink, et d'un noyau d'exécutif en C pour station de travail Unix et Linux. Les nouvelles fonctionnalités conduisant à augmenter de façon importante la complexité de l'heuristique d'optimisation de la distribution et de l'ordonnancement, un travail important de réécriture de cette dernière a dû être fait pour obtenir des performances équivalentes à celles de la version 5 sur des applications industrielles conséquentes comprenant plusieurs centaines de sommets pour le graphe d'algorithme et une quinzaine de sommets pour le graphe de l'architecture. La syntaxe des fichiers de sauvegarde des applications SynDEx (.sdx) ayant profondément changé, nous fournissons dans la distribution de SynDEx V6 un traducteur de V5 en V6. La plupart des applications réalisées avec SynDEx V5, dont plusieurs applications industrielles, ont été portées avec succès en V6. SynDEx V6.6 est actuellement téléchargeable à l'url : http://syndex.org.

Nous avons implanté dans SynDEx V6 l'algorithme optimal d'ordonnancement sans préemption avec contraintes multiples de précédences, de périodicités et de latences, d'un algorithme applicatif sur monoprocesseur.

Un noyau d'exécutif pour le processeur de traitement du signal TMS320C60 a été développé en collaboration avec le laboratoire ARTIST de l'INSA et le laboratoire des télécommunications de MITSUBISHI

ELECTRIC ITE. Ce noyau est écrit en C plutôt qu'en assembleur comme nous le faisions jusqu'à présent. Ses performances par rapport à l'assembleur étant bonnes, il pourra donc par la suite être adapté à tout autre processeur possédant un compilateur C, ce qui permettra un gain de temps important dans la réalisation de nouveaux noyaux.

7. Contrats industriels

7.1. ACOTRIS

Participants : Christophe Macabiau, Yves Sorel.

Le projet national RNTL ACOTRIS labellisé en 2001 comprend les partenaires suivants : CEA-LIST, CS-SI, INRIA-ESPRESSO-OSTRE, MBDA, SITIA. Il vise à proposer un environnement complet de spécifications des fonctionnalités, des architectures matérielles et des contraintes temps réel à l'aide d'une « extention temps réel embarquée de UML », permettant de faire de la vérification formelle avec le langage synchrone Signal (projet ESPRESSO de l'INRIA) et de l'implantation optimisée avec AAA/SynDEx d'applications distribuées temps réel embarquées. Nous avons terminé cette année l'application de référence d'appariement d'images en temps réel de MBDA réalisée avec SynDEX V6 sur une architecture multi-station de travail Unix. Nous avons participé à l'application de référence marine réalisée par CS-SI et SITIA. Nous avons participé à la réalisation d'une passerelle entre UML et Signal. Nous avons participé à la réalisation d'une interface entre Signal V4 et SynDEx V6.

7.2. MBDA

Participants : Christophe Macabiau, Julien Forget, Yves Sorel.

L'application de référence d'appariement d'images en temps réel, effectuée avec MBDA dans le cadre du projet ACOTRIS, nous a conduit à signer un contrat de collaboration pour l'aide à la conception avec AAA/SynDEx d'une nouvelle application industrielle de poursuite automatique de cible par traitement d'image multi-capteur. Cette dernière met en œuvre des algorithmes de traitement d'image en temps réel particulièrement complexes, utilisant des filtres de Kalman pour faire de la prédiction, et des modèles de Markov pour faire de l'estimation de mouvement.

7.3. ECLIPSE

Participants : Nicolas Pernet, Yves Sorel.

Le projet national RNTL ECLIPSE labellisé en 2002 comprend les partenaires suivants : CS-SI, INRIA-METALAU-OSTRE, PSA, SAPHIR-CONTROL, SITIA. Il vise à fournir un environnement logiciel libre de spécification/simulation/implantation optimisée d'algorithme de contrôle-commande fondé sur les logiciels libres Scilab/Scicos et SynDEx de l'INRIA. Nous travaillons avec le projet METALAU sur l'interfaçage entre Scicos et SynDEx, ce qui nous a conduit à revoir légèrement la sémantique de AAA/SynDEx sur les aspects rémanence des signaux, afin qu'elle soit cohérente avec celle de Scicos.

7.4. EAST-EEA

Participants : Frédéric Gager, Yves Sorel.

Le projet européen ITEA EAST-EEA labellisé en 2001 comprend les partenaires suivants : AB Volvo (S), AUDI AG (D), BMW AG (D), DaimlerChrysler AG (D), Centro Ricerche Fiat (I), Opel Powertrain GmbH (D), PSA Peugeot Citroen (F), Renault (F), Finmek Magneti Marelli Sistemi Elettronici (I), Robert Bosch GmbH (D), Siemens VDO Automotive AG (D), Siemens VDO Automotive S.A.S. (F), Valeo (F), ZF Friedrichshafen AG (D), ETAS GmbH (D), Siemens SBS - C-LAB (D), VECTOR Informatik (D), CEA-LIST (F), IRCCyN (F), INRIA-OSTRE-TICK-VERTECS (F), Linköping University of Technology (S), LORIA (F), Mälardalen University (S), Paderborn University - C-LAB (D), Royal Institute of Technology (S),Technical University of

Darmstadt (D). Il fait suite au projet national AEE qui comprenait les constructeurs et équipementiers Français, l'IRCCyN, le LORIA et l'INRIA et dont l'objet était les méthodes de conception d'architectures électroniques embarquée pour l'automobile. Nous avons principalement participé dans le cadre des lots 2 et 3 à la définition des exigences pour les applications distribuées temps réel embarquées de l'automobile, et à la définition d'un langage de description d'architecture couvrant tout le cycle de développement depuis l'architecture véhicule, jusqu'à l'architecture opérationnelle, en passant par les architectures fonctionnelles et matérielles. Ce langage fondé sur le standard UML 2.0 utilise les résultats obtenus dans le projet national AEE, à savoir le langage d'architecture AIL et le gestionnaire d'échange inter-composant (« middleware ». Nous devrions commencer l'année prochaine à travailler dans le cadre du lot 3 sur les aspects validation/vérification et sur les aspects implantation optimisée et génération automatique de code.

Nous collaborons avec l'action TICK et l'équipe SPORT du laboratoire I3S de Sophia Antipolis afin de mieux comprendre les liens qu'il y a entre les approches flot de contrôle utilisées dans UML0.2 pour décrire des comportements, et les approches flot de données utilisées pour décrire des calculs scientifiques. Dans ce but nous avons réalisé une interface entre le logiciel graphique de description de diagramme d'état SyncChart et SynDEx.

7.5. P2I

Participants : Julien Forget, Yves Sorel.

Le projet européen ITEA P2I labellisé en 2002 comprend les partenaires suivants : ESTEREL TECHNOLOGY (F), NOKIA Corporation (FN), THALES Communications (F), LIFL (F), INRIA-TICK-OSTRE (F), Tampere University of Technology (FN), University of Turku (FN). Il fait suite au projet RNRT PROMPT sur la programmation de SoC (System on Chip) pour les télécommunication qui s'est terminé en 2001. Il vise à proposer un environnement complet de spécifications des fonctionnalités, des architectures matérielles et des contraintes temps réel à l'aide d'une extention temps réel embarquée de UML, permettant de faire de la vérification avec les langages synchrones Esterel et Lustre qui cohabitent maintenant dans le logiciel Esterel Studio, et de l'implantation optimisée avec AAA/SynDEx d'applications distribuées temps réel embarquées. L'environnement sera testé sur des applications de télécommunications proposées par les partenaires industriels THALES-COMMUNICATION et NOKIA. L'environnement pourra à terme être industrialisé par ESTEREL-TECHNOLOGY.

8. Actions régionales, nationales et internationales

8.1. Actions nationales

8.1.1. Collaborations internes à l'INRIA

- BIP : nous continuons les travaux débutés dans l'ARC TOLERE sur les systèmes distribués temps réel embarqués tolérant aux fautes. Nous avons aussi des réflexions communes sur les problèmes d'ordonnancement régulé (« feed-back scheduling »).
- ESPRESSO : nous travaillons depuis de nombreuses années avec l'équipe à l'origine du langage synchrone Signal. Nous avions réalisé il y a quelques années une interface entre Signal H2 et SynDEx V4. Dans le cadre du projet RNTL ACOTRIS nous avons repris ces travaux afin d'une part d'être compatible avec les nouvelles versions de Signal et de SynDEx, et d'autre part d'approfondir la manière dont peuvent être traités dans AAA/SynDEx des programmes Signal multi-horloges (exochrones), plus particulièrement en essayant de comprendre l'impact de la notion d'« événement absent » dans Signal.

- IMARA : SynDEx est utilisé pour programmer et optimiser les applications s'exécutant en temps réel sur le véhicule électrique urbain CyCab conçu dans le projet IMARA. Ce véhicule possède une architecture multi-processeur comprenant de deux à quatre microcontrôleurs MPC555 et un PC embarqué sous Linux, connectés ensemble par un bus CAN. Nous visons à étendre nos techniques de programmation et d'optimisation à l'ensemble des véhicules développés chez IMARA. Pour cela nous participons à l'expérience PREVISU consistant à mettre à disposition aux visiteurs du site de l'INRIA Rocquencourt une flotte hétérogène de véhicules semi-autonomes, ainsi qu'aux projets européens CyberCars et Carsence.
- METALAU : nous avons co-encadré une thèse qui s'est terminée l'année dernière dont une partie du sujet consistait à produire du code temps réel distribué à partir de Scicos en utilisant AAA/SynDEx. Dans le cadre de cette thèse une version préliminaire d'une interface entre les deux logiciels libres de l'INRIA avait été réalisée. Cela nous a permis de participer au projet RNTL ECLIPSE dont le but est de proposer un environnement sans rupture de spécification/simulation/implantation fondé sur Scicos et SynDEx, permettant d'analyser et d'exploiter au niveau de la spécification (appelée aussi modélisation dans Scicos) les résultats obtenus en temps réel après implantation temps réel optimisée (SynDEx).
- SOSSO : évidemment depuis que notre équipe est séparée du projet SOSSO nous continuons à collaborer ensemble sur le sujet commun qui nous conduit à rester dans le même projet pendant de nombreuses années, à savoir, l'étude des liens entre l'automatique et l'informatique temps réel embarquée. Nous avons commencé à encadrer une thèse commune dans laquelle il s'agit de faire de l'ordonnancement adaptatif à travers une loi de commande (ordonnancement régulé) afin de traiter des problèmes mélangeant « temps réel strict » et « temps réel souple ». Nous envisageons de tester différentes stratégies d'ordonnancement adaptatif dans le cas d'applications de commande réalisées pour le CyCab.
- TICK : afin de préparer le projet ITEA P2I nous collaborons depuis un an en vue de fournir une interface efficace entre Esterel et SynDEx. Pour cela nous étudions une modification de la sémantique de notre modèle d'algorithme afin de pouvoir exploiter le résultat d'un nouveau compilateur d'Esterel fondé sur une structuration des programmes à l'aide de la hiérarchie des états, semblable à la hiérarchie des horloges de Signal, mettant en évidence les parties du programme qui sont actives et celles qui ne le sont pas, plutôt qu'un réseau d'automates mis à plat dans lequel cette information est perdue, comme c'est le cas pour la compilation de circuit. Ici encore nous sommes amenés à étudier ce que veut dire la notion d'« événement absent » du point de vue de AAA/SynDEx.

8.1.2. Collaborations avec d'autres laboratoires

- ESIEE COSI : nous cherchons principalement à analyser et apporter des solutions pour réduire la rupture qui existe entre la phase spécification/simulation de l'automaticien et la phase d'implantation à l'aide d'électronique numérique programmable et non programmable de l'informaticien et du concepteur de circuit. Pour cela nous étudions la robustesse d'algorithmes de commande relativement à des stratégies d'ordonnancement dans la lignée des travaux effectués à l'université de Lund en Suède. Nous continuons à travailler sur le simulateur, programmé en Scicos, du CyCab réalisé les années précédentes. Nous travaillons aussi à réaliser pour le CyCab des applications d'aide à la conduite à l'aide de capteurs à faibles coûts. Nous avons réalisé avec SynDEx V6 une application d'arrêt sur obstacle utilisant des capteurs ultrasons très simples, conçus par le laboratoire COSI. Nous étudions la possibilité de faire de l'évitement d'obstacles et du déplacement en train virtuel de CyCabs, en ajoutant aux capteurs ultrason des caméras à faibles coûts type webcam. L'objectif est de compenser la rusticité des capteurs par des traitements complexes distribués sur les MPC555.
- ESIEE A2SI : nous collaborons depuis plusieurs années afin d'étendre AAA/SynDEx à la

synthèse de circuit intégrés spécifiques. Une première thèse a conduit à un modèle d'implantation de circuit intégré à partir de la même spécification d'algorithme que celle utilisée pour les implantations multiprocesseur dans AAA/SynDEx. Ce modèle qui était incomplet car il ne traitait, pour les structures de contrôle, que l'équivalent en flot de données des boucles, a été étendu dans le cadre d'une seconde thèse, afin de traiter aussi le conditionnement, équivalent à la structure de contrôle « If...Then...Else ». Ceci permet de synthétiser complètement le chemin de contrôle et de données du circuit intégré, dont la fonctionnalité à été spécifiée sous la forme d'un graphe de dépendances de données conditionné factorisé. Cette synthèse automatique est dite optimisée car nous cherchons à l'aide d'heuristiques de trouver le meilleur compromis surface/latence=cadence à partir d'une caractérisation des opérations de l'algorithme relativement aux éléments de bibliothèque VHDL structurel synthétisable, dont nous nous servons pour produire le code « net-list » du circuit.

8.1.3. Collaboration avec ROBOSOFT

Participants : Frédéric Gager, Rémy Kocik, Yves Sorel.

Le CyCab est industrialisé par ROBOSOFT avec qui nous avons collaboré les années précédentes pour réaliser un noyau d'exécutif SynDEx V5 pour le microcontrôleur MPC555 et le PC embarqué sous Linux communicant par le bus CAN. Nous avons porté ce noyau ainsi que l'application de conduite manuelle du CyCab sur SynDEx V6, qui à terme va remplacer SynDEx V5. Chez ROBOSOFT les nouvelles générations de robots utilisent la même architecture matérielle extensible en nombre de processeurs utilisant des MPC555, et le PC embarqué sous Linux et communicant par le CAN. Nous coopérons avec eux afin de répondre aux besoins des nouvelles applications qui seront développées sur ces robots.

8.1.4. Collaboration avec l'INSA et MITSUBISHI ELECTRIC ITE

Participants : Thierry Grandpierre, Yves Sorel.

Nous collaborons depuis plusieurs années avec le laboratoire ARTIST de l'INSA de Rennes qui a réalisé une interface entre AVS, outil de spécification/simulation d'algorithmes de traitement d'image, et SynDEx. Dans le cadre de cette collaboration nous avions réalisé l'année dernière un noyau d'exécutif pour le processeur de traitement du signal TMS320C60. Ce dernier a été utilisé pour spécifier et implanter avec AVS/SynDEx des applications de codage/décodage d'image MPEG4 sur un réseau multi-TMS320C60. MITSUBISHI ELECTRIC ITE qui utilise le même processeur pour des applications de radio logicielle (software-radio), participe à ces travaux. Une collaboration tri-partite s'est établie visant à améliorer AAA/SynDEx pour des applications de traitement du signal et des images sur des architectures multi-TMS320C60 en tenant compte des particularités de la gestion de la mémoire de ce type de processeur. Cela a donné lieu au lancement d'une thèse co-encadrée.

Nous avons lancé une nouvelle collaboration avec le laboratoire LCST de l'INSA sur l'utilisation de SynDEx dans le cas multi-composant, où les composants programmables sont des TMS320C60 et les composants non programmables sont des FPGA. Afin que les deux types de composant puissent communiquer le laboratoire LCST a réalisé une IP capable, à l'intérieur d'un FPGA, de gérer des communications et des synchronisations suivant le même protocole SynDEx, que celui utilisé dans les processeurs sous la forme d'un programme réalisant un exécutif. MITSUBISHI ELECTRIC ITE qui envisage dans le cadre de sa plateforme de radio logicielle d'intégrer des FPGA, participe à cette collaboration.

8.2. Actions européennes

8.2.1. Réseau d'excellence IST ARTIST

Participant : Yves Sorel.

Afin de préparer les réseaux d'excellence du 6^{e} PCRD une initiative regroupant la plupart des acteurs académiques et industriels européens du domaine des systèmes temps réel critiques, nommée ARTIST

(Advanced Real-Time Systems), a été lancée. Elle a principalement consisté à établir dans ce domaine un état de l'art et un état des besoins à moyen terme (« road-map ») à la fois pour les cursus d'enseignement et pour les méthodes de conception industrielles de systèmes temps réel embarqués critiques.

9. Diffusion des résultats

9.1. Animation de la Communauté scientifique

Yves Sorel :

- est responsable du thème « Adéquation Algorithme Architecture » du PRC-GDR ISIS (Information Signal Images et viSion),
- participe au PRC-GDR ARP (Architecture, Réseaux et système, Parallélisme),
- participe à l'action spécifique SoC du CNRS.

9.2. Enseignement

Yves Sorel assure les cours :

- « Spécification, vérification et optimisation des systèmes distribués temps réel embarqués » à l'Ecole d'ingénieurs ESIEE de Noisy-le-Grand,
- « Adéquation algorithme-architecture » au DEA « Systèmes électroniques de traitement de l'information » d'Orsay.

9.3. Manifestations

Yves Sorel fait partie du comité scientifique des conférences EUSIPCO, AAA, GRESTSI, SYMPA, RTS.

10. Bibliographie

Bibliographie de référence

[1] L. CUCU, R. KOCIK, Y. SOREL. *Real-time scheduling for systems with precedence, periodicity and latency constraints.* in « 10th RTS2000 Real-Time Systems Conference », Paris, mars, 2002.

[2] A. DIAS, C. LAVARENNE, M. AKIL, Y. SOREL. *Optimized Implementation of Real-Time Image Processing Algorithms on Field Programmable Gate Arrays.* in « Fourth International Conference on Signal Processing », Beijing, China, octobre, 1998.

[3] A. GIRAULT, C. LAVARENNE, M. SIGHIREANU, Y. SOREL. *Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems.* in « 21st International Conference on Distributed Computing Systems, ICDCS'01 », Phoenix, USA, April, 2001.

[4] T. GRANDPIERRE. *Modélisation d'architectures parallèles hétérogènes pour la génération automatique d'exécutifs distribués temps réel optimisés.* thèse de doctorat, Université de Paris Sud, Spécialité électronique, 30/11/2000.

[5] T. GRANDPIERRE, C. LAVARENNE, Y. SOREL. *Modèle d'exécutif distribué temps réel pour SynDEx.* Rapport de Recherche, numéro 3476, INRIA, août, 1998, http://www.inria.fr/rrrt/rr-3476.html.

[6] T. GRANDPIERRE, C. LAVARENNE, Y. SOREL. *Optimized Rapid Prototyping For Real-Time Embedded Heterogeneous Multiprocessors.* in « CODES'99 7th International Workshop on Hardware/Software Co-Design », Rome, mai, 1999.

[7] C. LAVARENNE, O. SEGHROUCHNI, Y. SOREL, M. SORINE. *The SynDEx Software Environment for Real-Time Distributed Systems, Design and Implementation.* in « European Control Conf. », juillet, 1991.

[8] C. LAVARENNE, Y. SOREL. *Performance Optimization of Multiprocessor Real-Time Applications by Graph Transformations.* in « Conf. Parallel Computing », Grenoble, septembre, 1993.

[9] C. LAVARENNE, Y. SOREL. *Modèle unifié pour la conception conjointe logiciel-matériel.* in « Traitement du Signal », numéro 6, volume 14, 1997.

[10] Y. SOREL. *Massively Parallel Systems with Real Time Constraints, the "Algorithm Architecture Adequation Methodology".* in « Conf. on Massively Parallel Computing Systems », Ischia, Italy, mai, 1994.

[11] A. VICARD, Y. SOREL. *Formalization and Static Optimization for parallel implementations.* in « DAPSYS'98 Workshop on Distributed and Parallel Systems », septembre, 1998.

[12] A. VICARD. *Formalisation et optimisation des systèmes informatiques distribués temps réel embarqués.* thèse de doctorat, Université de Paris Nord, Spécialité informatique, 5/07/1999.

Communications à des congrès, colloques, etc.

[13] L. CUCU, R. KOCIK, Y. SOREL. *Real-time scheduling for systems with precedence, periodicity and latency constraints.* in « 10th RTS2000 Real-Time Systems Conference », Paris, mars, 2002.

[14] T. GRANDPIERRE, Y. SOREL. *Un nouveau modèle générique d'architecture hététogène pour la méthodologie AAA.* in « Journées Francophones sur l'Adéquation Algorithme Architecture », Monastir, Tunisia, décembre, 2002.

Bibliographie générale

[15] R. BALAKRISNAN, K. RANGANATHAN. *A Textbook of Graph Theory.* Springer, 2000.

[16] A. BENVENISTE, G. BERRY. *The Synchronous Approach to Reactive and Real-Time Systems.* in « Proceedings of the IEEE », numéro 9, volume 79, septembre, 1991, pages 1270-1282.

[17] S. BHATTACHARYA, K. MURTHY, E. LEE. *Software synthesis from dataflow graphs.* Kluwer Academic, 1996.

[18] L. CUCU, R. KOCIK, Y. SOREL. *Real-time scheduling for systems with precedence, periodicity and latency constraints.* in « 10th RTS2000 Real-Time Systems Conference », Paris, mars, 2002.

[19] J. DENNIS. *First Version of a Dataflow Procedure Language.* in « Lecture Notes in Computer Sci. », volume 19, Springer-Verlag, 1975, pages 362-376.

[20] F. GECSEG. *Products of automata.* série EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1986.

[21] N. HALBWACHS. *Synchronous programming of reactive systems.* Kluwer Academic Publishers, Dordrecht Boston, 1993.

[22] Z. LIU, C. CORROYER. *Effectiveness of heuristics and simulated annealing for the scheduling of concurrent task. An empirical comparison.* in « PARLE'93, 5th international PARLE conference, June 14-17 », pages 452-463, Munich, Germany, novembre, 1993.

[23] C. LIU, J. LAYLAND. *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.* in « Journal of the ACM », 1973.

[24] C. MEAD, L. CONWAY. *Introduction to VLSI systems.* Addison-Wesley, 1980.

[25] V. PRATT. *Modeling concurrency with partial orders.* in « International Journal of Parallel Programming », numéro 1, volume 15, 1986.

[26] A. ZOMAYA. *Parallel and distributed computing handbook.* McGraw-Hill, 1996.