Réalisation d'un système d'exploitation pour l'architecture reconfigurable dynamiquement OLLAF

Ismail Ktata



École Nationale d'Ingénieurs de Sfax

Cycle de Formation doctorale dans la discipline

Ingénierie des Systèmes Informatiques

&

Université de Cergy-Pontoise

Ecole Doctorale Sciences et Ingénierie

Spécialité: Sciences et Technologies de l'Information et de la Communication

THESE

En vue de l'obtention du

DOCTORAT

Par

Ismail KTATA

(Ingénieur Génie Electrique)

Réalisation d'un système d'exploitation pour l'architecture reconfigurable dynamiquement OLLAF

Soutenu le 21 Juin 2013, devant le jury composé de :

M.	**Nouri MASMOUDI**	*Président*
M.	**Abdellatif MTIBAA**	*Rapporteur*
M.	**Sébastien PILLEMENT**	*Rapporteur*
M.	**Mohamed ABID**	*Directeur*
M.	**Bertrand GRANADO**	*Directeur*
M.	**Fakhreddine GHAFFARI**	*Examinateur*

Dédicaces

A mon cher père et ma chère mère
pour toute la peine qu'ils se sont donné pour moi,
pour leur amour et leurs encouragements.
Le plaisir que j'ai de leur dédier ce travail n'arrivera nullement à compenser
leurs sacrifices qu'ils ont consentis pour m'aider à réussir.
Que Dieu vous garde en bonne santé et vous prête longue vie…

A ma femme qui m'a toujours encouragé,
acceptant tout ce temps soustrait à ma présence auprès d'elle,
Tu es une épouse exemplaire, ton affectation, ton aide et ta sympathie
sont l'essence de ma vie et le garent de ma réussite.
J'espère que tu trouve dans ce travail l'expression de mon amour sans limites.

A mon aimable fils AHMAD
pour le peu de temps que je lui ai consacré pendant mon travail.
Que Dieu te garde et te bénisse.

A mon frère, pour son amour et ses encouragements.
Qu'il puisse trouver ici tous mes sentiments de fraternité et fidélité.

A ma chère tante et sa famille qui m'ont été d'un grand soutien tout au long de
la période de ma thèse, que se soit en Tunisie mais surtout en France,

A mes beaux parents, mes beaux frères et ma belle sœur,

A toutes les familles KTATA & TURKI…

A mes amis…

A tous ceux que j'aime et qui me sont chers, je dédie ce travail
en témoignage de ma profonde gratitude et inestimable respect.

Remerciements

Je voudrais exprimer ma gratitude et mes remerciements les plus sincères à l'égard de toutes les personnes qui m'ont aidé aussi bien par leur soutient moral que par leur savoir et savoir-faire pour mener à bien ce travail :

Je voudrais remercier mes directeurs de thèse Mr. Mohamed ABID, Professeur à l'ENIS, et Mr. Bertrand GRANADO, Professeur à l'UCP, pour m'avoir accueilli au sein de leurs équipes de recherche CES & ETIS, pour m'avoir fait l'honneur d'encadrer mes travaux, pour leurs directives fructueuses et astucieuses, pour la confiance particulière qu'ils m'ont accordé, pour leur aides, leurs patiences et leurs dévouements durant toute la période de ma thèse. Je tiens à remercier aussi Mr Fakhreddine GHAFFARI pour son co-encadrement, ses nombreux conseils tout au long de cette thèse.

Mes remerciements s'adressent pareillement à Mr Nouri Masmoudi, Professeur à l'ENIS, pour l'intérêt qu'il a porté à ce travail en acceptant de me faire l'honneur de présider le jury de ma soutenance.

Je tiens également à remercier Mr Abdellatif MTIBAA, et Mr Sébastien Pillement d'avoir bien voulu me faire l'honneur d'être rapporteurs de ma thèse.

Mes remerciements distingués pour tous les chercheurs dans les deux laboratoires de recherche, en particulier : Samuel, Thomas, Guy, Amel, Jad, Yamen, côté ETIS ; Kais, Mossaad, côté CES, pour leur ambiance agréable, leur soutien et l'accueil chaleureux dont j'ai profité.
Ils ont été les meilleurs compagnons durant la période du projet et qui m'ont donné beaucoup de soutien par leurs idées inspirées et par leurs enthousiasmes communicatifs…

والعِلْمُ إنْ لم تَكْتَنِفْهُ شمائلٌ

تُعْلِيهِ كان مَطِيَّةَ الإخْفاقِ

لا تحسبنَّ العلْمَ ينفعُ وحْدَهُ

ما لم يُتَوَّجْ ربُّهُ بِخَلاقِ

حافظ إبراهيم

*Verse of the poet **Hafedh Ibrahim:**
Never think that science alone is a benefit, unless the owner is crowned by the morality

*Vers du poète **Hafedh Ibrahim :**
Ne pensez jamais que seule la science est un avantage, à moins que son propriétaire soit couronné par la morale.

Table des Matières

Liste des figures

Liste des tableaux

Glossaire

AAA	Adéquation Algorithme Architecture
ARD	Architecture Reconfigurable Dynamiquement
ASIC	Application Specific Integrated Circuit
CAD / CAO	Computer-Aided Design / Conception Assistée par Ordinateur
CCR	Central Configuration/Contexte Repository
CLB	Configurable Logic Block
CMU	Context Management Unit
DPGA	Dynamically Programmable Gate Arrays
DRLE	Dynamically Reconfigurable Logic Engine
DSP	Digital Signal Processing
EDF	Earliest Deadline First
FFT	Fast Fourier Transform
FGDRA	Fine Grained Dynamically Reconfigurable Architecture
FPGA	Field-Programmable Gate Array
GPP	General purpose processor
HCM	Hardware Configuration Manager
HW	Hardware
IP	Intellectual Property
LCM	Local Cache Memory
LE	Logic element
LLF	Least Laxity First
LUT	Look-Up-Table
MPEG	Moving Picture Experts Group
MPSoC	Multi-Processor System on Chip
OLLAF	Operating system enabled Low LAtency Fgdra
OS	Operating System (Système d'exploitation)
RISC	Reduced instruction set computer
RSoC	Reconfigurable System on Chip
RTOS	Real Time Operating System
SoC	System on Chip
SW	Software
WCET	Worst Case Execution Time

Chapitre I. Introduction générale

Insérés au sein des objets qui nous entourent, les circuits intégrés font de plus en plus partie de notre vie quotidienne, on les retrouve enfouis au sein de nos ordinateurs, nos téléphones portables, nos télévisions, nos voitures, et plus largement dans des domaines industriels tels que l'avionique, l'aérospatiale, l'automatisation, la distribution d'énergie, les télécommunications, etc… Cette évolution est la résultante d'une loi empirique, formulée par Gordon Moore [Moore, 1965] et du nom éponyme, qui prédit que la densité d'intégration de transistors par puce de silicium double tous les dix-huit mois. Cette évolution a permis d'intégrer des systèmes entiers sur une même puce en offrant aux applications s'exécutant en leur sein tous les composants qui leurs sont nécessaires. Dans la suite du manuscrit, ces systèmes seront appelés SoC, de leur acronyme anglais System on Chip. La conception de ces SoC est un véritable challenge scientifique et technique qui doit répondre à des impératifs applicatifs définis sous forme de contraintes : temps de calcul, consommation électrique, qualité de service, fiabilité, etc. Tout le challenge consiste à trouver les compromis technologiques et architecturaux permettant de satisfaire ces multiples contraintes. Plusieurs choix sont possibles, par exemple la conception de circuits dédiés sous forme d'ASIC développés pour réduire la consommation d'énergie induit par le fonctionnement du système. Mais cette conception doit aussi faire face à la contrainte du temps de mise sur le marché ou Time to market, pour laquelle une stratégie basée sur la réutilisabilité en utilisant des composants prédéfinis appelés bloc IP, pour Intellectual Property, est aujourd'hui largement plébiscitée. Ces composants sont utilisés comme des bibliothèques de blocs qui peuvent être assemblés et réutilisés dans le développement d'un circuit spécifique. Il est possible de regrouper ces IPs en deux classes, une première comprenant des IPs permettant d'exécuter du code logiciel et correspondant à des cœurs de processeurs (e.g. ARM, NIOS, LEON) et une seconde comprenant des IPs purement matérielles réalisant souvent des accélérateurs matériels (e.g. cœur IP FFT, cœur IP MPEG2, cœur IP H.264). Nous noterons Soft Core les IP de la première classe et IP matérielle les IP de la seconde classe.

Une tendance forte est de construire un SoC centré sur des Soft Core auxquels on associe des IP matérielles à l'aide d'une relation de dépendance maître-esclave. Cette méthode a l'avantage d'être bien maîtrisée et de pouvoir être rapidement mise en œuvre, puisqu'elle se base pour une grande part sur un flot de développement logiciel. En contre partie, elle lie très fortement les performances des SoCs à celles des cœurs de processeurs. Si nous nous référons à la comparaison entre l'évolution de la performance des processeurs et les besoins des applications exprimés par la complexité algorithmique (ou loi de Shannon), visible sur la

figure 1.1, on trouve une divergence croissante pénalisant les performances [Benoit et al., 2004]. Cet écart est lié au compromis flexibilité/performance réalisé qui limite les degrés de liberté dans l'exploration des solutions. Pour mieux répondre aux besoins des systèmes embarqués en termes de variété, performances et polyvalence (plusieurs applications dans un même système), il semble important de pouvoir disposer de toutes les IPs nécessaires au fonctionnement du système mais aussi d'envisager les différents modes d'interaction de ces IP en ne se limitant pas au mode maître-esclave et à une utilisation statique de ces IPs.

Une alternative intéressante pour répondre à ces limitations est l'utilisation d'architectures reconfigurables [Vassiliadis et al., 2007]. L'attractivité des SoC reconfigurables ou RSoC est induite par le compromis qu'ils offrent entre la performance et la flexibilité à mi-chemin entre une solution basée intégralement sur une architecture dédiée et optimisée et une architecture basée sur l'utilisation de cœur de processeurs.

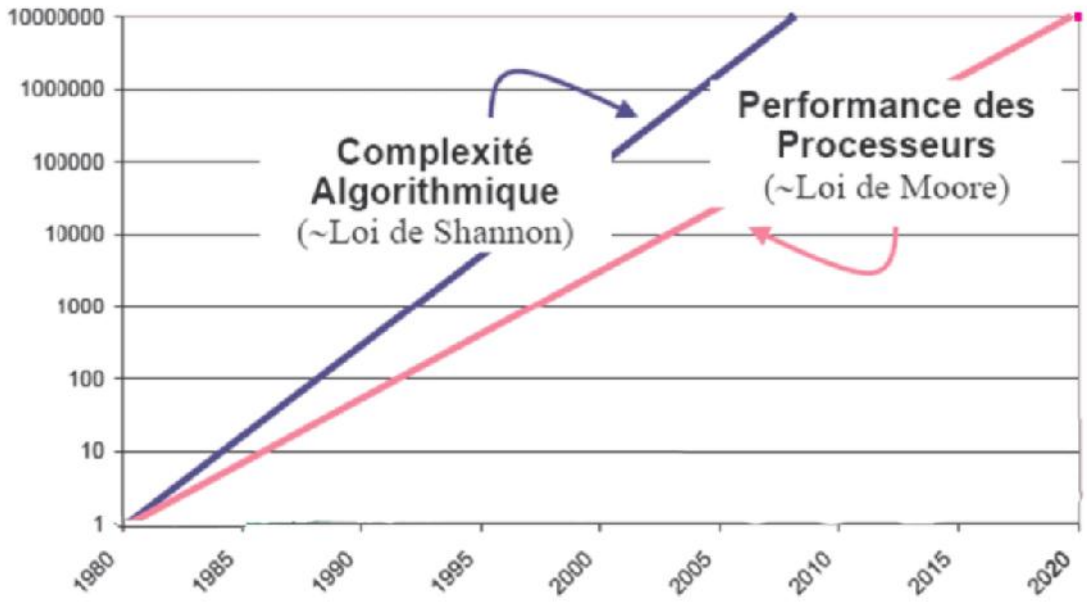

Figure 1.1. Evolution de la complexité algorithmique et de la performance des processeurs.

1. Contexte de l'étude

L'attrait des architectures reconfigurables réside, notamment, dans la possibilité qu'offrent ces architectures de répondre aux besoins de performances d'un système embarqué en utilisant des IPs correspondant à des accélérateurs matériels, tout en conservant, par l'intégration d'une

partie reconfigurable dans le SoC, la flexibilité des traitements que permet une machine programmable. Pour augmenter leur densité fonctionnelle, la reconfiguration de ces architectures peut-être dynamique, on parle alors d'architectures reconfigurables dynamiquement ou ARD. Il s'agit notamment de multiplexer spatialement et temporellement les différentes IPs matérielles réalisant les parties d'un algorithme sur la même surface logique correspondant à l'ARD à l'aide d'un ordonnancement spatio-temporel déterminé par les contraintes de l'application. Cette approche, bien que séduisante, est pénalisée par la complexité qu'elle apporte aux développeurs dans les différents niveaux du flot de conception. Cette complexité peut être réduite soit par le développement d'outils de CAO plus performants, soit en fournissant une couche intermédiaire, par exemple un système d'exploitation, dénoté OS de son acronyme anglais Operating System, qui permet d'abstraire cette complexité aux yeux de l'utilisateur. L'usage d'un OS est devenu une quasi-nécessité dans les systèmes embarqués qui exécutent des applications de plus en plus complexes et volumineuses avec de fortes contraintes temporelles, des limitations de ressources disponibles, tant en mémoire qu'en énergie disponible, mais également de la pression exercée par le marché sur ces produits. Le temps de développement doit être raisonnable, afin de limiter le temps de mise sur le marché, et ainsi d'assurer le succès du produit.

C'est au niveau OS pour les systèmes à base d'ARD que les travaux de cette thèse s'inscrivent. Ils s'appuient sur le projet SMILE (Système Mixte Intégré à faible Latence d'Exécution) qui vise à mettre en place un système sur puce comportant plusieurs unités de calculs de natures différentes (figure 1.2). Pour permettre de gérer efficacement la complexité de ce système, ces architectures seront gérées par un système d'exploitation distribué. Chaque unité de calculs, ou groupes d'unités de calculs de même type, dispose de son propre noyau optimisé pour la gestion de ce type de ressources tout en respectant un standard permettant une interopérabilité complète à plus haut niveau [Garcia et al., 2008].

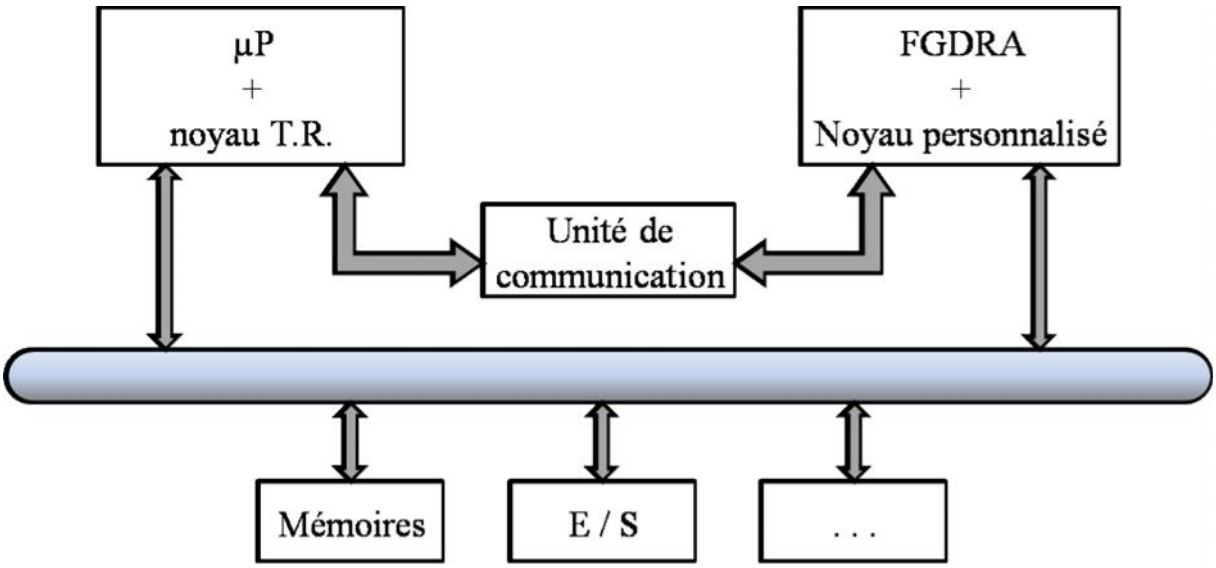

Figure 1.2. Projet SMILE

2. Problématiques de l'étude

Afin de gérer la complexité d'un RSoC, il est possible d'avoir recours à un OS inspiré de ce qui est fait dans les systèmes purement logiciels. Dans notre travail, nous nous intéressons à définir des services originaux d'un OS pour gérer efficacement une ARD conçue pour supporter efficacement un OS et appelée OLLAF [Garcia, 2012].

Les problématiques soulevées dans cette étude sont les suivantes :

- un point clé de nos travaux de recherche est de pouvoir utiliser les ARD indépendamment d'un processeur. C'est pour cela, que l'architecture OLLAF a été conçue [Garcia et al., 2008]. Cette architecture permet de supporter efficacement des services d'OS conçus pour sa gestion, à travers une hiérarchie de mémoire de configuration, des unités de gestion automatiques des contextes, ainsi qu'un média de communication et d'accès aux entrées/sorties rationalisé. Nous contribuons par ces travaux de thèse au développement de ces services d'OS spécifiques pour l'architecture OLLAF.

- Un second point abordé dans ces travaux est la spécification des propriétés dynamiques d'une application qui présentent des caractéristiques aléatoires et peu prédictibles. Cette spécification doit permettre aux services d'OS de mieux gérer les caractéristiques dynamiques de l'application.

- Un dernier problème abordé dans le cadre de ces travaux est la méthodologie de déploiement d'une application dynamique sur OLLAF.

3. Contributions

Les contributions de nos travaux de thèse se résument en trois points majeurs :

1. L'élaboration d'un modèle de représentation des applications très dynamiques présentant des caractéristiques non-déterministes et aléatoires. Ce modèle est paramétrique et flexible pour pouvoir s'adapter à une application donnée.

2. La proposition d'une approche efficace pour implémenter une application sur OLLAF.

3. Le développement d'un ordonnancement permettant l'implémentation de l'approche proposée et la tester sur des applications temps réel.

4. Organisation du manuscrit

Le manuscrit est organisé selon le plan suivant :

- Chapitre 2 : nous présentons le contexte de nos travaux qui abordent le domaine des ARDs. Nous présentons les motivations qui nous ont conduit à adopter l'architecture OLLAF pour le traitement des applications dynamiques. Nous étudions et comparons différents systèmes d'exploitation proposées pour les ARDs. À partir de cette étude, nous positionnons nos travaux et nous donnons les grandes lignes de nos contributions.

- Chapitre 3 : notre contribution à la modélisation d'une application dynamique est présentée. Une vue globale du modèle générique est présentée. Nous détaillons alors ses caractéristiques, ses aspects paramétriques et ses différents composants.

- Chapitre 4 : la démarche de l'approche d'ordonnancement prédictif proposée est détaillée. À ce niveau, nous présentons les différentes étapes proposées dans l'approche ainsi qu'une comparaison des techniques de prédictions testées.

- Chapitre 5 : la mise en œuvre de l'approche depuis la génération du modèle jusqu'à la génération de l'ordonnancement d'exécution est décrite. Des études de cas sur des applications multimédia sont présentées. Ces études expérimentales permettent de valider nos contributions méthodologiques et d'ordonnancement.

- Chapitre 6 : nous concluons cette thèse par le bilan des travaux effectués avant d'aborder quelques perspectives à nos travaux.

Chapitre II. Reconfiguration dynamique et ordonnancement temps réel : État de l'art

1. Introduction

Ce deuxième chapitre est consacré à l'étude des architectures reconfigurables. Le chapitre commence par la définition des architectures reconfigurables : leurs caractéristiques, leurs limites. Il présente ensuite un état de l'art sur ces architectures, en présentant notamment l'architecture OLLAF. La dernière section est consacrée pour présenter un état de l'art sur l'ordonnancement et en particulier celui dans le cadre des architectures reconfigurables dynamiquement.

2. Les architectures reconfigurables

Dans le cadre de l'électronique numérique et des SoC, une architecture peut-être définie comme un ensemble d'éléments de traitement, de mémorisation et d'une topologie de connexion reliant entre eux ces éléments [Bossuet, 2004]. Il est possible d'adjoindre à une architecture la propriété de reconfiguration, c'est à dire la possibilité pour certains de ses composants matériels de modifier leur fonctionnalité dans le but de s'adapter aux besoins applicatifs. Une architecture possédant la propriété de reconfiguration ou architecture reconfigurable est une architecture qui a la possibilité de supporter plusieurs configurations lui permettant de changer matériellement son état de fonctionnement au court du temps. Cette reconfiguration peut affecter tous les éléments constituant l'architecture.

Durant ces vingt dernières années les architectures reconfigurables ont été à la source d'une importante révolution technologique. Elles se sont imposées comme une réelle alternative à la conception de circuits ASICs. Parmi les architectures reconfigurables les FPGAs [Brown et al., 1992] sont les représentants les plus visibles. Cette révolution technologique est due d'une part aux inconvénients liés à la conception des ASICs que ce soit au niveau du temps de conception qu'au niveau du coût des outils de conception, et d'autre part au compromis performance/flexibilité qu'apportent les architectures reconfigurables [Pillement, 2010]. Ce compromis est illustré par la figure 2.1, sur laquelle sont représentés différents choix de conception entre les processeurs généraux, ou GPP, les processeurs spécialisés, les architectures reconfigurables et les ASIC suivant deux axes un pour la performance et un pour la flexibilité. Un ASIC est une architecture statique conçue spécifiquement pour une application. Il est optimisé pour l'application cible et présente un degré de flexibilité quasi nul. A l'opposé, on trouve les architectures programmables

(processeurs généraux et spécialisés) qui du fait de leur programmabilité ont un grand degré de flexibilité qu'ils payent au détriment des performances. Les architectures reconfigurables sont un compromis entre flexibilité et performance. Elles permettent de mettre en œuvre un design relativement optimisé pour une application donnée proche d'un design ASIC, tout en gardant grâce à la reconfiguration dynamique une bonne flexibilité s'approchant de celles des processeurs. Elles présentent aujourd'hui une solution intéressante pour développer des systèmes embarqués flexibles et à haut degré de performances. Il est encore possible d'obtenir plus de flexibilité si l'architecture est reconfigurée dynamiquement pour s'adapter aux besoins de l'application qu'elle exécute au fil de l'eau [David, 2003].

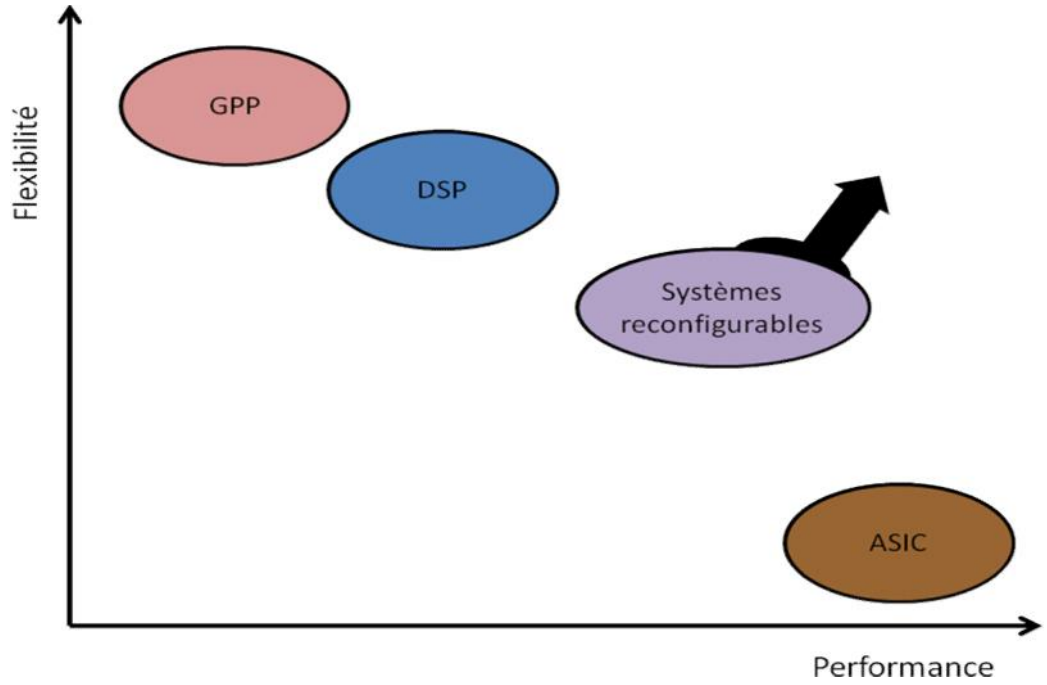

Figure 2.1. Compromis Flexibilité/Performances de différents types d'architectures

2.1. La reconfiguration dynamique

La reconfiguration dynamique a pour objectif d'améliorer l'efficacité d'une architecture en permettant l'allocation et la réutilisation de ses ressources pour plusieurs tâches. Elle permet de changer la configuration, et donc la fonctionnalité, d'une partie du circuit sans avoir besoin de stopper le fonctionnement de tout le circuit. Les ARD ont des caractéristiques parmi lesquelles la durée de configuration et d'exécution des tâches qui sont primordiales [Lallet, 2008]. Leurs capacités sont influencées par certains choix [Lallet, 2008] parmi lesquels :

- Le type d'implémentation algorithmique qui peut être en trois modes : implémentation temporelle (logicielle), spatiale (matérielle) ou spatio-temporelle (reconfigurable). Ces trois méthodes sont différentes du point de vue temps d'exécution, consommation et surface de silicium nécessaires à l'exécution de l'algorithme. Pour une implémentation logicielle, le passage d'une configuration à une autre est plus rapide que celle matérielle qui, en revanche, est plus performante en temps d'exécution.

- Le choix de la granularité de la reconfiguration : une architecture reconfigurable à grain fin permet une grande flexibilité mais au prix d'un grand flot de données de configuration, et donc, d'une longue durée de reconfiguration. En revanche, le fait de travailler sur des mots de plusieurs bits permet d'avoir des périodes de configuration moins longues, mais restreint les choix de configuration au re-routage de l'interconnexion d'opérateurs gros grains [Compton et Hauck, 2002].

- Le choix de la méthode de reconfiguration : une reconfiguration dynamique mono-contexte présente un seul plan ou contexte de configuration avec la possibilité de reconfigurer les parties de l'architecture du circuit qui ont changé d'état. Une reconfiguration dynamique multi-contextes nécessite de stocker dans des mémoires locales, plusieurs plans de configuration. Durant l'exécution de l'application et selon les besoins de traitement, un contexte est sélectionné pour être chargé et configuré sur l'architecture. Cette dernière méthode peut accélérer les phases de reconfiguration mais nécessite plus de ressources de mémorisation.

- Le choix du réseau d'interconnexion: les réseaux d'interconnexion servent à assurer les communications entre les différentes unités de traitement. On trouve plusieurs topologies d'interconnexion : globaux, point-à-point, hiérarchiques. Le choix de la topologie a des impacts sur les performances d'exécution, les temps de reconfiguration et la consommation [David et al, 2005].

Une multitude de solutions architecturales ont été proposées dans l'industrie ainsi que dans les laboratoires de recherche [Bossuet, 2004]. Ces propositions essaient de garantir le meilleur compromis entre deux solutions extrêmes :

- une qui est flexible et capable d'implémenter n'importe quelle application, et

- l'autre qui n'est pas flexible mais performante pour une application donnée.

2.2. Différents niveaux de reconfiguration

Une des caractéristiques principales des architectures reconfigurables est la granularité de la reconfiguration. Il s'agit de la complexité des blocs reconfigurables constituant l'architecture, ou encore, « la taille en bit de la donnée élémentaire manipulée » [Ben Abdallah, 2009]. Cette caractéristique de granularité varie du plus petit au plus gros. On peut distinguer essentiellement deux niveaux : logique et fonctionnel [David, 2003]. Des exemples d'architectures reconfigurables de différentes granularités feront l'objet des sous sections suivantes. Un état de l'art plus complet est présenté dans les travaux suivants [Garcia, 2012], [Vassiliadis et al., 2007], [Abel, 2006] et [Bossuet, 2004].

2.2.1. Architecture reconfigurable au niveau logique

Cette reconfiguration est appelée aussi la reconfiguration à grain fin. La taille de l'élément reconfigurable de base est une unité logique simple travaillant au niveau du bit (LUT, bascules, porte logiques, ...). Les LUTs (Look-Up-Table) sont des mémoires considérées comme des tables de vérité programmables qui peuvent réaliser n'importe quelles fonctions booléennes des entrées. C'est ce qui offre à ce type d'architecture le moyen d'avoir une grande flexibilité pour s'adapter aux mieux aux spécifications de l'application en exécution. L'architecture la plus représentative de l'approche à grain fin est le FPGA qui domine aujourd'hui le marché. Ces architectures ont prouvé leur utilité pour plusieurs domaines d'applications demandant un calcul intensif comme le traitement du signal et des images, en assurant un traitement accéléré parfois de plusieurs ordres de grandeur par rapport à une solution programmée sur processeur.

Un FPGA est formé de trois composants :

- Les éléments logiques configurables ou encore (CLB) pour la famille Xilinx et (LE) pour la famille Altera. Ils sont les éléments de calcul configurables.
- Un réseau d'interconnexion configurable reliant les éléments de calcul. On trouve plusieurs topologies de connexion.
- Les blocs d'entrées/sorties qui connectent le FPGA avec les périphériques externes. Ils servent entre autre à télécharger les configurations à partir d'une mémoire externe.

Parmi cette famille d'architectures reconfigurables à grain fin, on retient :

a) L'ATMEL AT40K

L'AT40K [ATMEL, 2002] a été l'une des premières architectures dotée de la reconfiguration partielle. Cette architecture est formée d'une matrice parfaitement régulière de blocs logiques identiques. Chaque élément logique est constitué de deux LUT à trois entrées et d'une bascule D (figure 2.2) [ATMEL, 2013]. La flexibilité de reconfiguration est maximale. On peut reconfigurer à la volée à un grain inférieur au LE en ne modifiant par exemple que le contenu d'une des LUT pour modifier le traitement. Les blocs sont interconnectés entre eux par des interconnexions directes (liaisons horizontales, verticales et diagonales) et à travers un réseau.

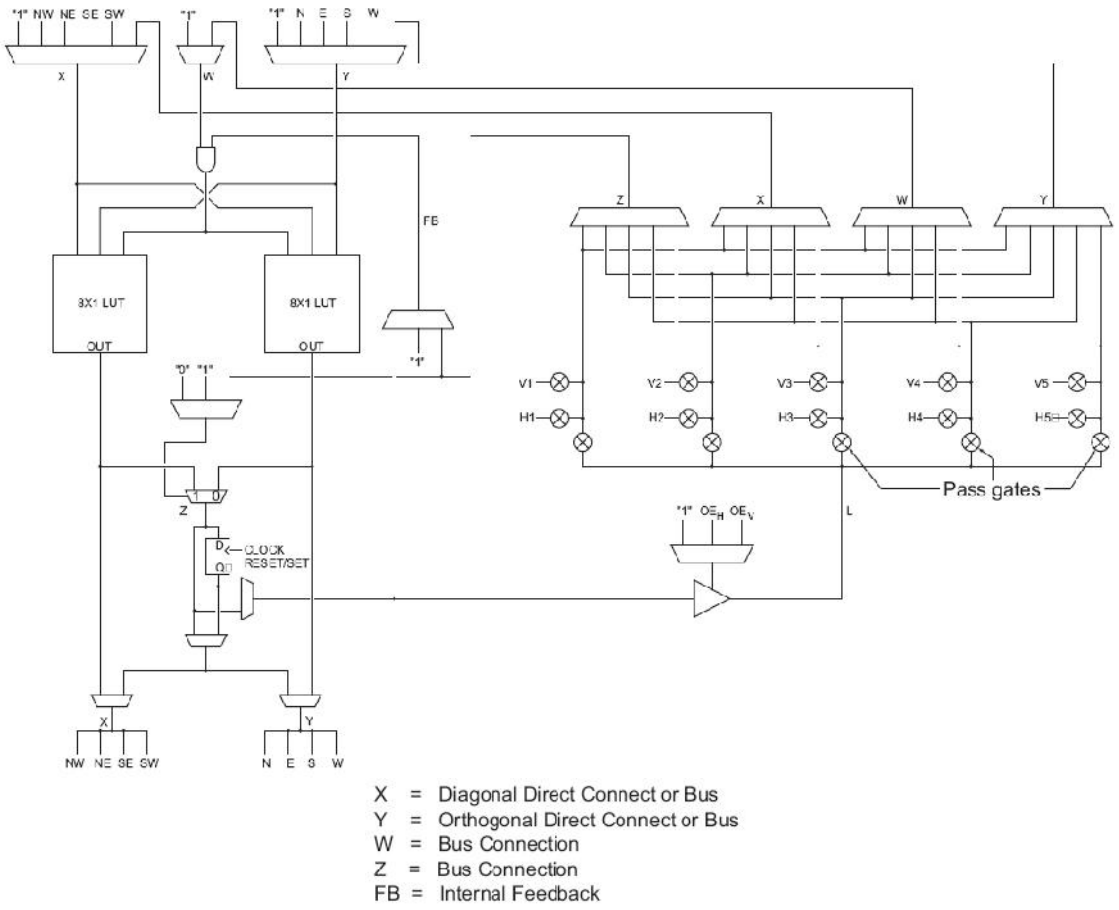

Figure 2.2. Bloc logique de l'AT40K

Les FPGAs AT40K ont été utilisés comme architectures de base dans le projet ARDOISE [Abel, 2006]. Ce projet est la collaboration entre une dizaine de laboratoires français pour définir une ARD constituée de 3 à 8 FPGAs AT40K. Chaque FPGA est intégré dans une carte fille et toutes les cartes filles sont interconnectées à une carte mère. La carte mère, appelée aussi contrôleur de configuration, est celle responsable du séquencement des reconfigurations des cartes filles.

b) Famille Virtex de XILINX

La famille Virtex des FPGA Xilinx [Xilinx, 2001] est très répandue. Elle se base sur une topologie de type matrice hiérarchique. Cette matrice est à base de CLBs formé chacun de deux *slices* et tel que chaque *slice* groupe deux blocs logiques (figure 2.3). Contrairement à l'AT40K, le bloc logique de Virtex est composé de quatre LUTs à quatre entrées (ou six à partir du Virtex 5). Les CLBs sont interconnectés à travers une matrice de routage. D'autre ressources logiques sont également présents (multiplexeurs, bloc RAM, etc.) pour accélérer le calcul. Le grain de reconfiguration diffère selon la version de la famille considérée. Dans les Virtex et VirtexIIpro, la reconfiguration s'effectue par tranche couvrant la hauteur totale de la matrice de logique reconfigurable et d'une largeur de 4 CLB. Dans les Virtex 4, 5 et 6 la reconfiguration s'effectue par frame d'une largeur d'un seul CLB et de hauteur respectivement 16, 20 et 40 CLB. Généralement, l'interface interne utilisée pour la reconfiguration dynamique se présente sous la forme d'un port parallèle appelé ICAP, pour Internal Configuration Access Port, de 32 bits pouvant fonctionner à une fréquence maximale de 100 MHz. D'autres interfaces de configuration, présentées dans [Xilinx, 2013], sont possibles mais présentent une bande passante au mieux égale à l'ICAP si l'on utilise les outils Xilinx. Certains travaux universitaires permettent de doubler les performances, c'est le cas des travaux autour de l'IP Farm [Duhem et al., 2011] et de l'IP UPaRC [Bonamy et al., 2012].

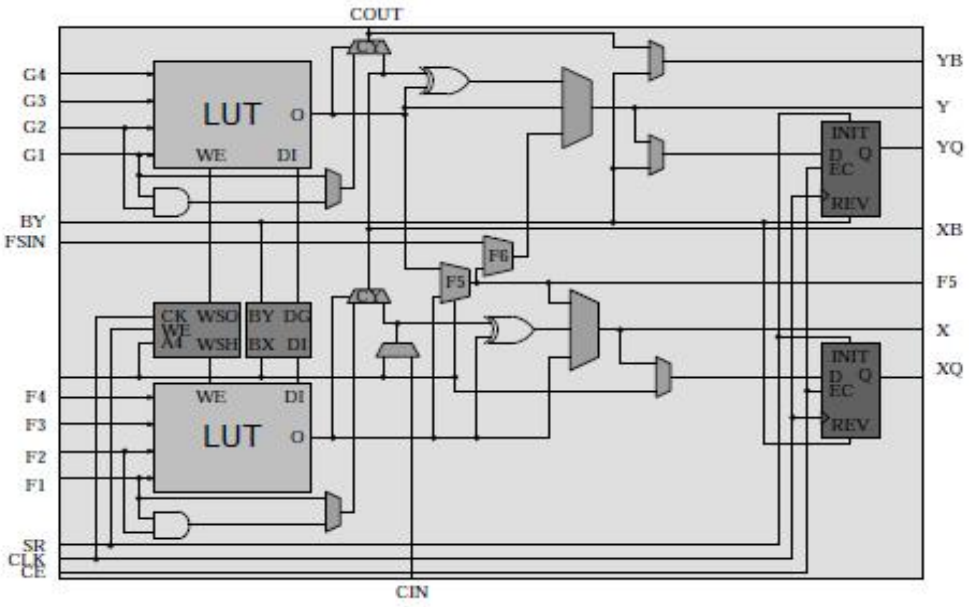

Figure 2.3. Bloc logique de type Virtex

Les architectures de type grain fin présentent certains inconvénients. D'une part, la complexité importante du placement-routage : pour réaliser de grandes unités de calcul il faut interconnecter un grand nombre d'éléments logiques à travers un grand nombre de commutateurs dédiant la majorité de la surface du circuit au routage [Zhuo et al., 2006]. D'autre part, et d'après le guide utilisateur consacré à la reconfiguration dynamique du constructeur [Xilinx, 2011], les temps de reconfiguration, pour les Virtex 4,5 et 6, sont liés à la taille du volume de données de l'IP à configurer et à la bande passante de l'interface. Si le système fait recours à plusieurs reconfigurations pendant la durée du traitement, ceci va nécessiter un très grand volume de données et un temps et une énergie non négligeable, ce qui peut dégrader la performance. Par exemple, la configuration d'une seule LUT à quatre entrées d'un FPGA nécessite seize bits de configuration auxquels s'ajoutent des bits de configuration des interconnexions ou encore de calculs de retenue. Si on considère l'architecture XCV50 de Xilinx comme le plus petit composant de la famille Virtex, son *bitstream*[1] nécessite 559200 bits pour une reconfiguration complète. Ce chiffre peut atteindre facilement des dizaines/centaines de millions des bits pour un circuit de la famille Virtex 6/Virtex 7 [Xilinx, 2013].

c) Famille Stratix d'ALTERA

Le Stratix-III [Altera, 2011] comme le Stratix-II [Altera, 2007] dont elle a hérité de nombreuses caractéristiques, a deux niveaux de hiérarchie. Le niveau le plus haut, consiste en un ensemble d'éléments configurables appelé LAB (Logic Array Bloc). Tout comme les CLB des FPGA Virtex, ils sont répartis en matrice. A ce même niveau, des mémoires de différentes tailles sont réparties sur la matrice, ainsi que des blocs dits « DSP ». Ces derniers permettent de disposer de 48 à 384 multiplieurs (18bits x 18bits). Selon le mode de configuration, ces blocs DSP peuvent fonctionner en multiplieur, MAC (Multiplication ACcumulation) ou additionneur de deux/quatre multiplieurs. Au niveau inférieur les LAB sont constitués de 8 ALM (Adaptive Logic Modules) et d'un réseau de connexions locales. Cette structure est donc très proche de celle du Virtex-IV avec les CLB et les « slices ». Cependant les ALM ont une architecture qui évolue par rapport aux LE (Logic Element) qui sont les briques reconfigurables élémentaires des anciennes générations d'architectures Altera. Les ALM (figure 2.4) sont réalisés autour d'un bloc de logique combinatoire à 8 entrées, de deux additionneurs et des registres de sortie. Le bloc combinatoire est réalisé avec deux LUT à

[1] Ensemble des bits permettant la configuration d'un circuit reconfigurable

quatre entrées et de quatre LUT à trois entrées. Parmi les innovations de Stratix-V [Altera, 2013], on trouve l'ajout de registres au sein de l'architecture ALM, des blocs mémoires de 20 Kbits contre 9 Kbits auparavant et des blocs DSP à précision variable.

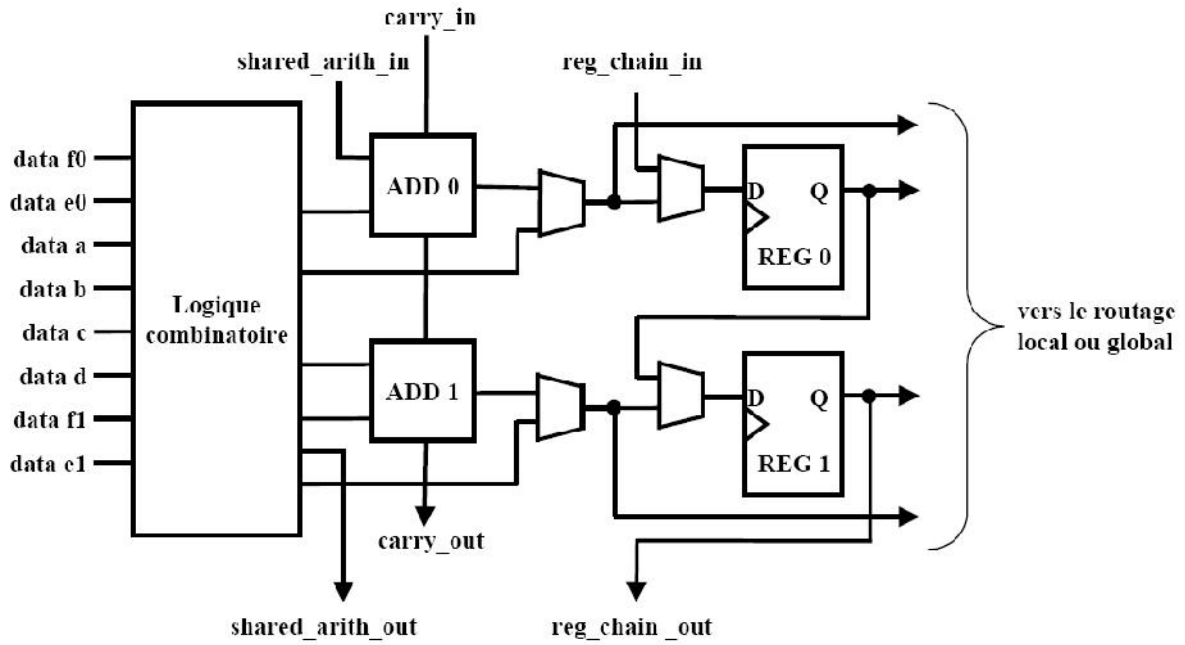

Figure 2.4. Architecture d'un ALM du Stratix-III.

Les architectures à grain fin offrent une grande flexibilité et peuvent être utilisées pour mettre en œuvre, théoriquement, n'importe quel circuit numérique. Toutefois, en raison de la configuration à grain fin, ces systèmes présentent des performances moyennes, des surcoûts de configuration élevés, et faible taux d'utilisation de la surface reconfigurable. Diverses techniques ont été utilisées pour réduire le temps de latence de reconfiguration, comme le préchargement et la mise en cache de configuration. Les techniques de préchargement assurent la réduction de la latence en permettant la reconfiguration en pipeline des opérations de reconfiguration et d'exécution. L'exemple de l'IP FaRM combine le préchargement à une méthode de compression des contextes de configuration [Duhem et al., 2011]. Généralement, le préchargement nécessite de connaître à l'avance la configuration suivante, tandis que la technique de mise en cache nécessite tout simplement la connaissance des reconfigurations les plus communes et souvent nécessaires, afin qu'elles puissent être stockées dans la mémoire cache. De nombreuses études portent aujourd'hui sur l'amélioration des mémorisations, des technologies nouvelles apparaissent et promettent des évolutions importantes. Parmi celles potentiellement intéressantes pour le domaine des architectures reconfigurables on peut citer

les MRAM (Magnetoresistive RAM) [Torres et al., 2011]. Cette technologie remplace les DRAM par un aimant en matériau ferromagnétique. La direction du champ magnétique peut changer et occuper une ou deux positions stables. Le grand avantage de ces mémoires est qu'elles peuvent conserver les informations mémorisées en l'absence de source d'énergie. Cependant le champ magnétique à appliquer doit être important afin de changer le moment magnétique du point de mémorisation. Ceci implique une forte consommation de puissance ainsi qu'une grande quantité de chaleur dissipée.

2.2.2. Architecture reconfigurable au niveau fonctionnel

Ce niveau de reconfiguration, nommé aussi reconfiguration à gros grain ou à grain « épais », offre une solution qui permet d'optimiser le processus de reconfiguration en diminuant la taille des bitstreams. Dans ce type d'architecture, les éléments de base sont des UAL, des cœurs de processeurs et des opérateurs arithmétiques (additionneurs, multiplieurs). Généralement, les architectures à gros grain ciblent un domaine d'application spécifique [Hartenstein, 2001]. La nature des blocs matériels implémentés est basée sur les besoins relatifs à ce domaine. La flexibilité est obtenue par l'organisation de ces blocs sur la surface de l'architecture de façon à pouvoir les interconnecter de la façon la plus efficace pour réaliser la fonctionnalité demandée.

Les architectures reconfigurables au niveau fonctionnel s'approchent des ASICs dans leur performance du fait de leur spécialisation, et s'approchent des GPP de point de vue flexibilité puisqu'elles sont reconfigurables [Cardoso and Hübner, 2011]. Ces architectures sont plus faciles à configurer qu'une architecture à grain fin. La taille de leur bitstream est diminuée par rapport aux architectures à grain fin ce qui influe directement sur le temps de reconfiguration et la consommation d'énergie [Amano, 2006]. Bien que les architectures à gros grain offrent de bonnes performances, elles souffrent d'un manque de flexibilité qui freine leur développement et la réalisation de circuits commerciaux les intégrant.

Dans la suite, nous présentons quelques architectures reconfigurables à gros grain.

a) L'architecture MorphoSys

Cette architecture, proposée dans [Parizi et al., 2002], est à base de cellules reconfigurables. Chaque cellule reconfigurable est formée de quatre éléments de base : un UAL, un bloc d'interconnexion d'E/S, une mémoire qui sauvegarde l'ensemble des configurations possibles

de l'architecture et un bloc de composant logique à grain fin. Elle est formée d'une grille de 8x8 cellules.

b) L'architecture DART

C'est une architecture à reconfiguration dynamique conçue pour traiter les applications de télécommunication mobiles de troisième génération [David et al., 2002]. Le but de l'architecture est d'optimiser son efficacité énergétique. L'adaptation de la plateforme est effectuée d'une manière partielle, en ligne et dynamique. Cette architecture est composée d'un contrôleur de tâches, de ressources de mémorisation et de ressources de calcul appelées clusters. Un cluster (figure 2.5) est composé de datapaths reconfigurables (DPR) interconnectés au travers d'un réseau de bus segmentés. A ce niveau, se retrouve la mémoire de configuration ainsi que son contrôleur. Le contrôleur de tâches affecte les différents traitements à exécuter aux différents clusters. Chaque cluster intègre un cœur de traitement dédié et six DPR opérant sur des données de largeur allant de 8 à 32 bits.

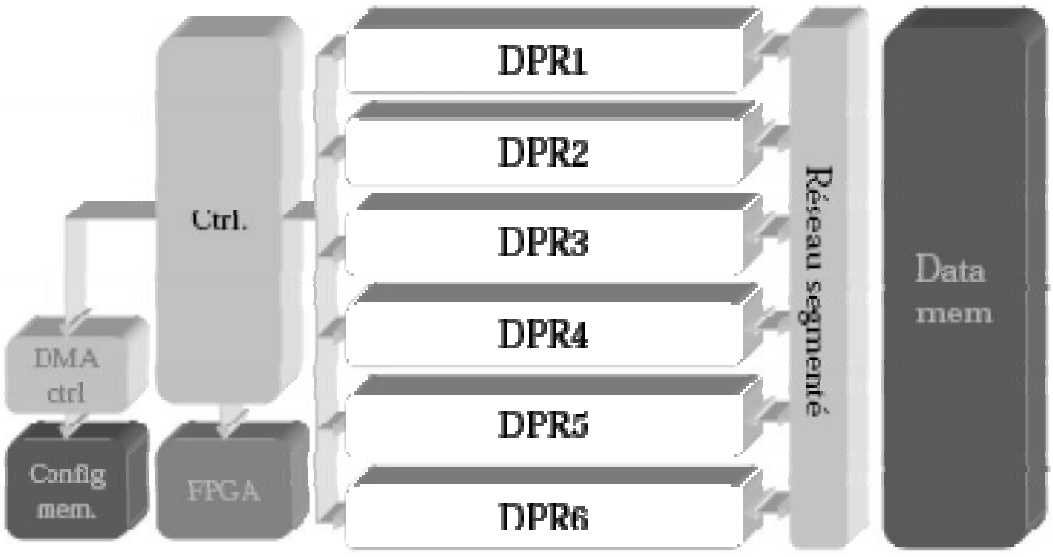

Figure 2.5. Architecture d'un cluster de DART

c) Systolic Ring

L'architecture Systolic Ring [Sassatelli et al., 2002] est une architecture à reconfiguration dynamique partielle, dédiée aux traitement flots de données. L'unité de calcul de base de cette architecture est le Dnode (figure 2.6). Il peut être assimilés à un petit microprocesseur de 16 bits, avec un chemin de donnée configurable (à grain épais) construit autour d'une Unité Arithmétique et Logique câblée (Multiplieur/Additionneur) travaillant sur des mots de 16 bits

stockés dans un banc de registres de 4 x 16 bits. Chaque Dnode a un contrôleur local qui possède une mémoire de configuration pouvant stocker jusqu'à huit instructions. La reconfiguration se fait au niveau cluster, qui est l'assemblement de deux Dnodes. Chaque cluster peut être reconfiguré en un cycle à la fréquence de fonctionnement. En plus des contrôleurs intégrés à chaque Dnode, Systolic Ring dispose d'un contrôleur de configuration global. Il s'agit d'un processeur RISC avec un jeu d'instructions hybride pour être capable de contrôler la structure reconfigurable.

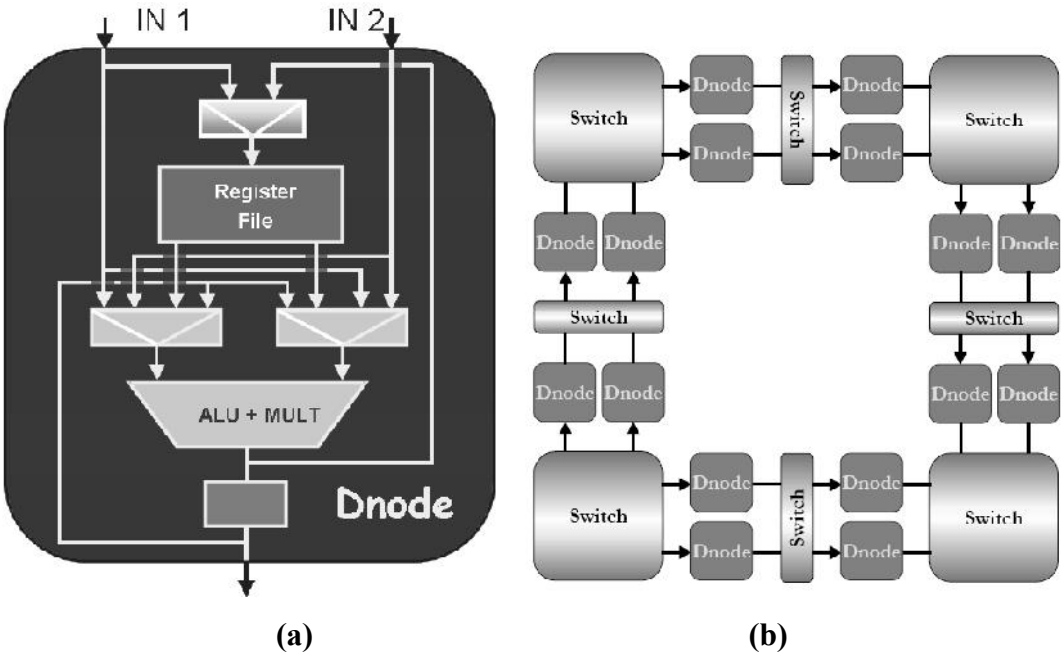

Figure 2.6. L'architecture Systolic Ring: (a) Le Dnode, (b) Couche opérative

d) eXtreme Processing Platform

XPP [Baumgarte et al., 2003] est une architecture reconfigurable dynamiquement à grain épais mixte composée de matrice d'unités de traitement prédéfinies appelées « *Processing Array Elements* ». Ces éléments peuvent être des unités UAL, des mémoires (FIFO, RAM) ou des LUTs. Cette architecture (figure 2.7) est conçue pour supporter différents types de parallélisme, comme le multithreading. La reconfiguration est initiée par un contrôleur global appelé « Supervising Configuration Manager » qui envoie les configurations vers les différents « Processing Array Elements ». Chacune de ces sous-matrices dispose d'un contrôleur de configuration local qui peut soit reconfigurer directement les « Processing Elements » via un schéma séquentiel de propagation en arbre, soit sauvegarder la configuration reçue dans une mémoire cache locale.

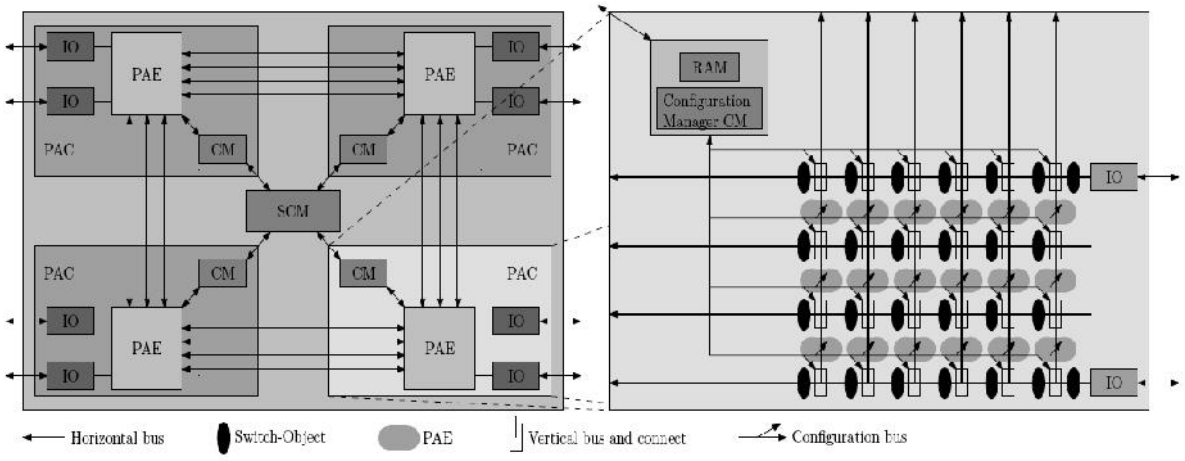

Figure 2.7. L'architecture XPP

Bien que les architectures reconfigurables permettent d'avoir à la fois une performance de traitement élevée et une grande flexibilité, leur utilisation est limitée par leur complexité de gestion. Pour faire face à cette complexité et permettre la mise en œuvre rapide d'une application sur une architecture reconfigurable, une solution consiste à masquer cette complexité en ayant recours à un OS. Cette voie a donné lieu à de nombreux travaux de recherches, dont nous présentons ici un sous-ensemble.

3. Les systèmes d'exploitation pour les systèmes reconfigurables

Un RSoC, est un système complexe reconfigurable intégré sur une seule puce. Il contient des processeurs, de la mémoire, des ARD et des périphériques externes [Chang et al., 1999]. La complexité d'un tel système nécessite l'utilisation d'un système d'exploitation pour garantir une gestion efficace des tâches tout au long de l'exécution de l'application. Le système d'exploitation utilisé, doit supporter la reconfiguration dynamique. Plusieurs travaux de recherche ont été consacrés à la définition de nouveaux OS adaptés aux RSoC. Nous présentons les principaux systèmes d'exploitation existants dans la littérature et conçus pour la gestion des architectures reconfigurables.

3.1. MARC.1

Wigley et Kearney, ont présenté dans [Wigley et Kearney, 2001] et [Wigley and Kearney, 2002] un OS appelé MARC.1 (en anglais : Management of Application for Reconfigurable Computing) qui est considéré comme le premier noyau supportant l'exploitation de la reconfiguration dynamique. Ce système d'exploitation reçoit l'application matérielle sous forme de graphe de tâches qu'il doit exécuter sur un FPGA. Les nœuds du graphe de tâches sont regroupés en fonction de leur temps au plus tôt (As Soon As Possible), de façon à maximiser le nombre de nœuds qui peuvent être exécutés simultanément. Pour le placement, la solution est construite progressivement en plaçant un nœud du graphe de tâche à la fois, conformément à un ensemble de règles. Le but est de garantir que les tâches qui sont liées dans le graphe par un arc seront placées dans des blocs logiques contigus pour simplifier le routage. Comme applications de test, les auteurs utilisent une partie du codeur MP3 et un algorithme de reconnaissance de cible.

3.2. ReConfigME

Dans [Wigley et al., 2006], les auteurs ont proposé quelques améliorations ajoutées au premier prototype MARC.1 pour donner naissance à un OS appelé ReConfigME. ReConfigME est un système d'exploitation multicouche qui fournit une interface de client pour l'utilisateur et qui exige une description matérielle de l'application qui sera allouée sur un FPGA. La description de l'application est supposée être fournie sous forme de graphe de tâche et chaque nœud du graphe doit être synthétisé dans un format intermédiaire de type EDIF. Le système d'exploitation est organisé en trois couches qui communiquent en utilisant un protocole TCP/IP standard. Les trois couches sont :

- La couche utilisateur, qui présente une interface entre l'utilisateur et l'OS. Elle permet à l'utilisateur de faire circuler des données des entrées/sorties depuis ou vers la plateforme.

- La couche OS, qui reçoit le graphe des tâches de l'application et le partitionne. L'application est divisée de façon à ce que toutes les partitions puissent être allouées sur l'espace disponible du circuit FPGA. Si tous les emplacements pour les partitions de l'application ont été trouvés, le placement et le routage seront effectués pour

générer un code de configuration pour programmer le FPGA. Sinon, l'application sera bloquée et mise en file d'attente en raison du manque d'espace disponible.

– La couche plateforme, qui reçoit le bitstream à partir de la couche OS pour l'implémenter sur le FPGA.

Pour ajouter une nouvelle application, l'exécution sur le circuit FPGA sera interrompue et une nouvelle configuration contenant la nouvelle et l'ancienne application sera générée et implémentée. Ainsi, l'ancienne application poursuit son exécution avec la nouvelle application. Cette approche introduit un surcoût considérable du temps d'exécution pour l'application.

3.3. OS pour plateformes embarquées reconfigurables

Dans [Steiger et al., 2004], les auteurs présentent un système d'exploitation pour les environnements temps réel, tout en mettant l'accent sur le problème d'ordonnancement. Dans la définition proposée, une application matérielle n'est pas partitionnée lors de l'exécution (divisée en parties qui peuvent être intégrées dans une même surface d'exécution), et le système d'exploitation ne gère que les modules pré-placés et pré-acheminés appelés tâches, dont la surface en nombre de ressources de calculs exigée, et le temps d'exécution, sont connus à priori. Toutefois, la tâche est disponible sous une forme indépendante de la position, ce qui fait que le code de configuration correspondant doit être généré lors de l'exécution, en fonction de l'emplacement qui est attribué par le système d'exploitation. Aucune information n'est fournie sur le surcoût qui est introduit par la génération de la configuration lors de l'exécution. L'OS est formé de deux partie : une partie logicielle qui assure la génération de la configuration et effectue l'ordonnancement et le placement des tâches. Pour l'ordonnanceur, cela évite le ré-ordonnancement des tâches qui étaient déjà garantis. Le placement est effectué par le gestionnaire de ressources, qui assure le suivi des ressources allouées. Une autre partie, la partie matérielle de l'OS assure la configuration du circuit, ainsi que d'autres services typiques tels que la communication interprocessus, la gestion de la mémoire et l'accès aux ports d'entrées/sorties.

3.4. OS4RS

Dans [Nollet et al., 2003], du laboratoire IMEC en Belgique, une extension du système d'exploitation RTAI, Real Time Application Interface [RTAI] basée sur le noyau Linux, est proposée. L'approche, appelée OS4RS, se concentre principalement sur le développement d'une infrastructure de communication uniforme pour les tâches logicielles et matérielles, et sur l'ordonnancement et la migration des tâches. L'infrastructure de communication est utilisée pour mettre en œuvre la communication interprocessus, tandis que la politique d'ordonnancement est conçue de manière à équilibrer la charge de travail entre les processeurs et les FPGAs. La migration des tâches est possible dans OS4RS. Une tâche peut être déplacée d'un GPP à un autre au cours de son exécution, ou elle peut migrer d'un processeur vers une des tuiles reconfigurables du circuit FPGA. La migration d'un processeur à un autre processeur différent ainsi que d'un processeur vers un FPGA ne peut se faire qu'à des instants précis appelé « checkpoints » ou points de contrôle. Lors d'un checkpoint, une tâche qui a été déjà choisie pour être migrée, enregistre son contexte d'exécution pour pouvoir reprendre son exécution sur une autre architecture. On notera que dans cette mise en œuvre, plus de la moitié du temps de calcul du processeur est utilisé pour gérer uniquement les communications.

3.5. BORPH

Dans [So et al., 2006], les auteurs ont proposé une extension appelée BORPH[2] du noyau Linux. BORPH définit une tâche matérielle en tant qu'une IP exécutable sur le FPGA. Une tâche matérielle est gérée par l'OS à la manière d'une tâche logicielle exécutée sur un processeur. Une tâche matérielle est encapsulée dans un fichier exécutable spécifique appelé BORPH Object File ou BOF. Ce fichier comprend des informations sur l'emplacement de la tâche matérielle, décidé lors de la conception, et le bitstream de configuration de l'IP. Lorsque l'application, contenant les tâches matérielles, est exécutée, elle est gérée par l'OS comme une application purement logicielle qui peut accéder à tous les services fournis par le noyau. La configuration d'une tâche matérielle est encapsulée dans le fichier BOF, et est pré-synthétisée pour une position spécifique sur le circuit reconfigurable. Dés lors, on ne peut pas décider à l'exécution le placement le plus efficace pour la tâche ce qui limite l'exploitation de la

[2] Berkeley Operating system for ReProgrammable Hardware

reconfiguration du circuit. La solution bien qu'attrayante manque de portabilité, une grande partie du noyau étant profondément liée aux spécificités de l'architecture reconfigurable.

3.6. Discussion

Les OS discutés dans cette section sont de deux types. Les OS du premier type sont développés adhoc pour optimiser les performances : optimiser l'utilisation de l'espace reconfigurable, réaliser le partitionnement, générer le bitstream de configuration. Le deuxième type d'OS modifie un noyau existant, souvent linux, offrant ainsi plus de flexibilité pour l'utilisateur, mais moins d'efficacité dans la gestion des ressources. A ce jour, les travaux portant sur le développement d'OS pour les ARD présentent des limitations :

- ils se basent généralement sur un modèle de développement inadapté et sont fortement liés aux possibilités limitées offertes par les plates-formes de la famille Virtex de Xilinx ;

- ils utilisent les théories existantes fondées sur des systèmes purement logiciels, fermant un certain nombre de possibilités intéressantes qui pourraient être offertes par la reconfiguration dynamique.

D'autre part, si nous analysons les architectures reconfigurables dynamiquement présentées au paragraphe II.1 aucune de ces architectures ne prend en compte l'OS dans sa conception. Cela conduit à des choix architecturaux rendant certains services d'OS peu efficaces. Ces plateformes présentant généralement des temps de configuration et des coûts de préemption limitant fortement la dynamicité du système, et rendant difficile un ordonnancement préemptif et temps réel efficace. La prise en compte de problématiques d'OS lors de la conception d'une plateforme reconfigurable est nécessaire pour pouvoir offrir aux développeurs d'application une solution complète, fonctionnelle, efficace et réellement utilisable.

Dans ce cadre, et pour pouvoir offrir pleinement les caractères de versatilité et de dynamicité aux systèmes utilisant des architectures reconfigurables dynamiquement à grain fin, appelées encore FGDRA, nous avons défini dans notre équipe de recherche l'architecture OLLAF (**O**perating system enabled **L**ow **LA**tency **F**gdra). Cette plateforme est spécialement

conçue pour supporter et améliorer l'efficacité des services de système d'exploitation nécessaires à la gestion d'une ARD [Garcia et al., 2009].

Une description plus détaillée de cette architecture est donnée dans le paragraphe suivant.

4. OLLAF : nouvelle ARD

Les FPGAs disponibles sur le marché bien que présentant une possibilité de reconfiguration dynamique sont aujourd'hui plus optimisés pour une exécution optimale d'applications statiques, et n'arrivent que partiellement à atteindre l'ensemble des objectifs de la reconfiguration dynamique qui sont les suivants :

- la rapidité d'exécution et la satisfaction des contraintes temps réel,
- la bonne gestion des ressources,
- la grande flexibilité,
- l'efficacité du processus de configuration.

C'est pour cela, que notre équipe a conçu une architecture innovante de type FGDRA permettant de mieux satisfaire ces objectifs. Cette architecture, appelée OLLAF [Garcia et al., 2009], a été proposée pour répondre à certaines problématiques parmi lesquelles :

- l'amélioration de la vitesse de chargement des tâches. Il s'agit de minimiser les coûts temporels des transferts des bitstream de configuration. Ce coût représente la majeure partie du surplus temporel dû à l'OS. Ceci est encore plus important si on considère un système dont la gestion des tâches est dynamique, puisque ces transferts peuvent avoir lieu dans le pire cas à chaque instant où le système d'exploitation est appelé.
- La préemption de tâches matérielles. Il s'agit de proposer une gestion matérielle de la sauvegarde et de la restauration de contexte d'exécution des tâches. Là encore, l'accent est mis sur la réduction du surplus temporel dû aux transferts.
- Le problème de gestion des ressources de calcul. Ceci reprend à la fois la gestion temporelle, à travers l'ordonnancement, et la gestion spatiale à travers l'allocation des

ressources. Le but est de pouvoir simplifier au maximum ce problème déjà très complexe en garantissant la possibilité de déplacer une tâche à la fois temporellement mais aussi spatialement sans nécessiter de traitement lourd et sans altérer la fonctionnalité ou les performances de la tâche.

L'architecture OLLAF a été conçue pour répondre aux forts besoins de dynamicité présents dans les applications de type robots domestiques, dispositifs intelligents dans l'automobile, télé-chirurgie, circuits intelligents implantés dans le corps humain, réseaux de capteurs intelligents, etc. [Duranton et al., 2009]. L'exécution de ces applications nécessite une plate-forme capable de supporter en temps réel leur dynamicité et leur complexité. Pour cela OLLAF est spécialement conçue pour améliorer l'efficacité des services d'OS nécessaires à sa gestion [Garcia et al., 2009]. A l'égard des autres propositions d'architectures spécifiques pour la reconfiguration dynamique [Compton et Hauck, 2000] [Shibata et al., 2000], l'ensemble de l'architecture OLLAF, visible sur la figure 2.8, est conçue pour fonctionner en symbiose avec un système d'exploitation permettant de mieux gérer des tâches matérielles dans un système préemptif.

Dans cette architecture le cœur logique reconfigurable est découpé en plusieurs blocs parfaitement identiques et de taille définie, appelés tranches. Chaque tranche est formée d'un double plan de configuration réalisant ainsi un FGDRA multi-contextes. Chaque tâche utilise un certain nombre de tranches contiguës. Lorsqu'un des plans d'une tranche, appelé plan actif, exécute une tâche, alors le second, appelé plan caché, est en mode sauvegarde/restauration d'une autre tâche. On peut ainsi, parallèlement à une exécution sur le plan actif, effectuer un transfert entre la mémoire contenant les contextes de configurations et/ou d'exécution et le plan caché. Pour améliorer la gestion des contextes et des configurations, OLLAF possède des mémoires caches associées à chaque tranche. Ces mémoires peuvent contenir chacune un petit nombre de contextes de données et de configurations. Pour résoudre les problèmes d'accès concurrents à la mémoire globale, chaque tranche a été munie de contrôleurs de gestion de contexte d'exécution et de configuration.

4.1. Définitions et caractéristiques

Chaque tranche de l'architecture OLLAF peut être reconfigurée séparément et offre les mêmes services. Pour s'exécuter, une tâche doit allouer un nombre entier de tranches contiguës. Une telle topologie limite le problème d'ordonnancement à un problème à deux

dimensions : temps + surface 1D. En outre, l'uniformité des éléments de calculs dans OLLAF facilite la migration des tâches d'une tranche à une autre sans avoir besoin de modifier les données de configuration. Chaque tranche est constituée d'un certain nombre d'éléments logiques appelés LE. Ces LE sont répartis selon une matrice régulière. Une tranche est connectée aux deux tranches qui lui sont contigües à travers deux ports d'entrée/sorties. Un autre port d'entrées/sorties relie la tranche au média de communication. La version actuelle d'OLLAF se base sur des tranches contenant 512 LE.

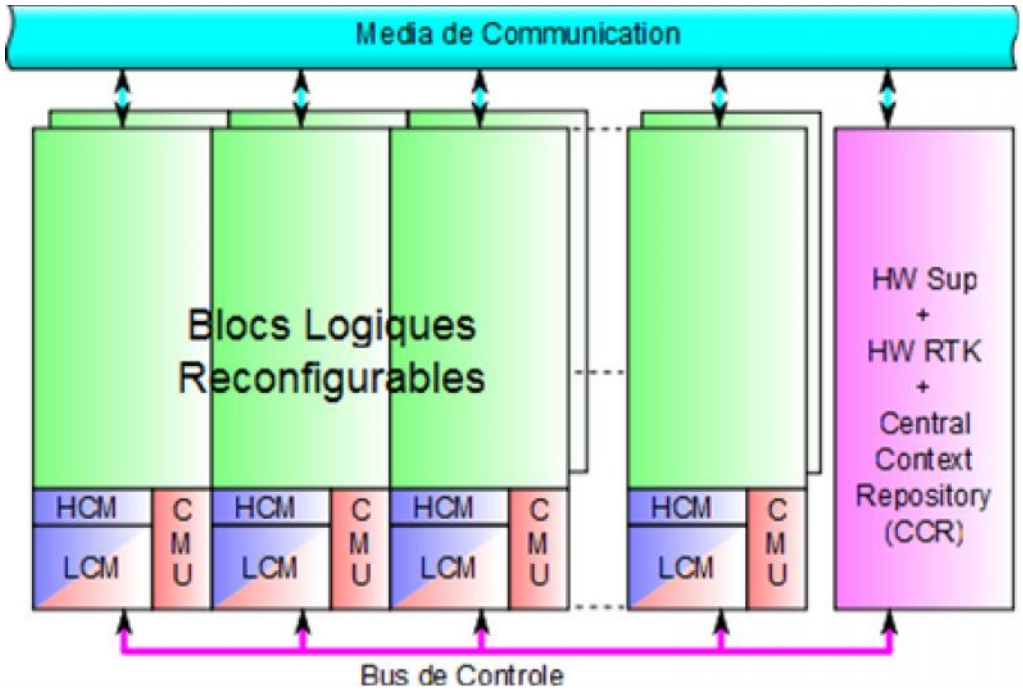

Figure 2.8. Vue globale de l'architecture OLLAF

Les éléments logiques, visibles sur la figure 2.9 [Garcia, 2012], sont similaires aux éléments logiques utilisés classiquement dans les FPGAs. Ils sont composés d'une LUT à 4 entrées, d'une bascule D, de multiplexeurs et d'inverseurs programmables.

Dans chaque tranche il y a une mémoire locale et deux contrôleurs locaux autonomes : le CMU et le HCM. Le CMU, pour Context Management Unit, est une IP matérielle capable de gérer automatiquement les sauvegardes et restaurations des contextes d'exécution. Le HCM, pour Hardware Configuration Manager, est une IP matérielle qui a pour fonction de transférer une configuration depuis la mémoire locale vers le cœur logique reconfigurable. Les transferts de contexte d'exécution et de configuration sont assurés via des scanpaths qui sont connectés respectivement aux deux contrôleurs. Les transferts entre le cœur de la logique reconfigurable

et les mémoires locales peuvent s'effectuer parallèlement sur toutes les tranches. Cela permet de réduire les effets de goulot d'étranglement au niveau de la mémoire de configuration.

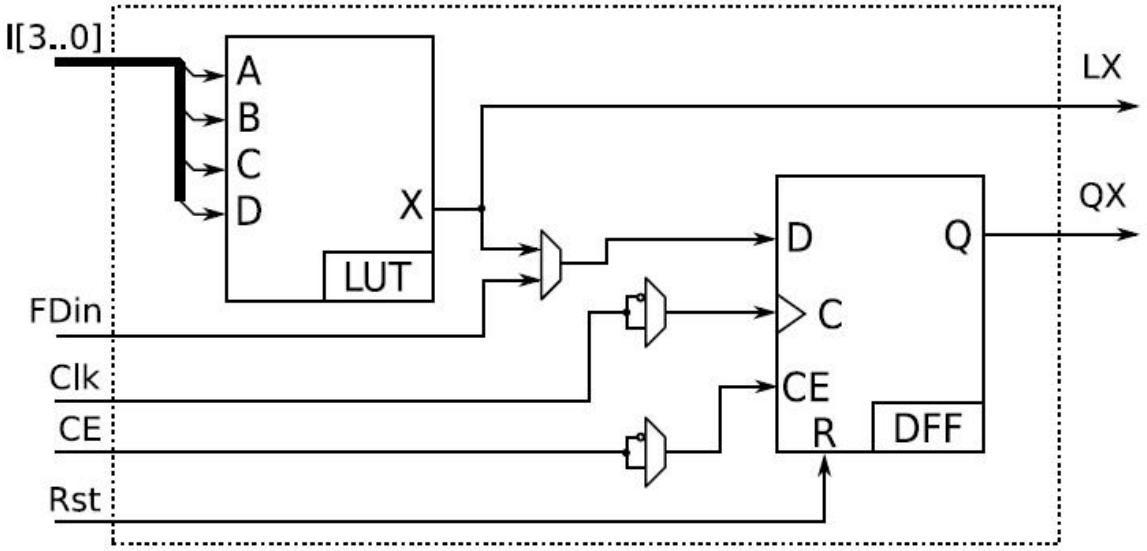

Figure 2.9. Vue fonctionnelle d'un élément logique dans OLLAF

La mémoire locale appelée LCM, pour Local Cache Memory, est composée de deux blocs mémoire distincts : l'un permet de sauvegarder les contextes de données d'exécution et l'autre sauvegarde les configurations. Chaque LCM permet de sauvegarder trois contextes et trois configurations, soit trois tâches complètes. Le bitstream complet d'une telle tranche, comportant à la fois la configuration et le contexte d'exécution de la tranche, comporte 21 Koctets.

Visible sur la figure 2.8, un bloc représentant le superviseur est nommé HW Sup. Il peut exécuter un noyau temps réel. Il est relié au bus de contrôle qui connecte : une mémoire globale appelée CCR, pour Central Context Repository, les LCM et les contrôleurs locaux, HCM et CMU. La mémoire CCR représente le niveau de mémoire le plus haut. Elle sauvegarde tous les contextes de l'application qui sera exécutée sur le cœur reconfigurable d'OLLAF.

Toutes les entrées/sorties ainsi que les accès aux mémoires de données s'effectuent via un média de communication extérieur au cœur reconfigurable. Cela permet par exemple de laisser au système les problèmes de gestion d'accès multiples aux ressources. Cela doit aussi permettre une communication entre tâches indépendantes non seulement de l'emplacement où

les tâches sont exécutées mais également du statut des dites tâches (Idle : pour dire qu'un processus est en repos - Busy : lorsqu'il s'agit d'une ressource occupée - Delayed : lorsqu'une tâche dépasse le délai - Release : libérer une ressource - Awake : tâche réveillée). Par exemple, une tâche peut envoyer des informations à une autre sans avoir à se soucier de savoir où se trouve la tâche destinataire, ni même si cette tâche destinataire est réellement en train de s'exécuter à cet instant précis.

L'architecture proposée permet d'améliorer significativement l'efficacité d'un certain nombre de services d'un système d'exploitation :

- La préemption : si une tâche de basse priorité est en exécution et une autre plus prioritaire arrive, la première sera interrompue et les ressources de calculs seront allouées à la nouvelle tâche. Cela nécessite une commutation des contextes : la sauvegarde du contexte de la tâche de basse priorité et la restauration/chargement du contexte de la tâche de haute priorité.

- La relocalisation des tâches : après avoir été préemptée, une tâche peut reprendre son exécution dans un autre emplacement. Elle doit être donc reconfigurée dans le nouvel emplacement et doit charger son contexte de données déjà sauvegardé lors de la dernière préemption.

- La configuration des tâches : assurer la configuration d'une tâche tout en optimisant la vitesse de configuration.

- L'ordonnancement des tâches bénéficiant directement des possibilités offertes par les trois services précédents.

- Les communications inter-tâches,

4.2. Gestion des contextes

Le chemin de configuration dans OLLAF est particulièrement adapté pour améliorer l'efficacité des chargements de configuration ainsi que la gestion des contextes d'exécution lors de la préemption des tâches. Ce chemin de configuration peut être vu comme une hiérarchie de mémoire visible sur la figure 2.10. Le niveau le plus bas est composé des deux plans, exécution et caché, du cœur logique reconfigurable. Chaque plan étant coupé en deux

parties, l'une représentant les informations de configuration de l'application, l'autre regroupant le contenu des différentes bascules de la tranche ou, en d'autre termes, le contexte d'exécution. Les contextes sont gérés par l'unité de gestion des contextes CMU qui permet les transferts entre les mémoires cache LCM et les tranches, réalisant ainsi le second niveau de mémoire. La version actuelle d'OLLAF permet d'effectuer soit une sauvegarde de contexte soit une restauration de contexte, soit les deux à la fois.

Cette structure complexe de transferts de configuration, comportant plusieurs niveaux de mémoire et différents contrôleurs matériels permettant de gérer les transferts de façon automatique, montre tout son intérêt si on est capable de pré-charger les tâches avant que leur exécution ne soit demandée par l'ordonnanceur. Il est pour cela nécessaire d'utiliser un ordonnanceur prédictif qui doit permettre de pré-charger les tâches avant que leur exécution effective ne soit demandée par l'ordonnanceur principal. Cela a son importance ici car avec cette technique il est possible de placer une même tâche à plusieurs endroits différents. L'ordonnanceur principal décidera alors en temps réel d'exécuter cette tâche à un de ces emplacements. Dans ce cas, si cette tâche est préemptée par la suite, il y aura alors deux contextes différents d'une même instance de tâche placés dans deux mémoires locales différentes, sans oublier le contexte déjà placé dans le dépôt central de contexte CCR, dernier niveau de la hiérarchie mémoire, qui sera bien sûr un contexte erroné après la préemption. Il s'agit alors d'un problème de cohérence de contexte. Afin de garder une cohérence après chaque sauvegarde et restauration, il faut associer à chaque instance de tâche un numéro de version à son contexte d'exécution. Lorsque la tâche est préemptée, son contexte est sauvegardé dans la mémoire locale et le numéro de version associé est incrémenté. Ce numéro de version doit être immédiatement récupéré par le superviseur. Lorsqu'on veut reprendre cette tâche, le superviseur envoie en plus de l'ordre de chargement de la tâche, le numéro de version courant du contexte de l'instance de tâche (une instance de tâche est une occurrence d'exécution de la tâche). Avant d'effectuer le transfert, le CMU vérifie que le contexte présent dans la mémoire locale présente bien le même numéro de version. Dans le cas contraire il génère une interruption au niveau du superviseur afin qu'un rechargement complet du contexte à jour soit effectué. En attendant que ce transfert s'effectue, la ressource reste alors disponible pour, par exemple, continuer l'exécution de la tâche en cours d'exécution sur la tranche concernée.

Pour illustrer le fonctionnement des tranches dans OLLAF, la figure 2.11 présente deux tâches, T1 et T2, qui sont exécutées sur la même tranche. T1 étant en exécution sur le plan

actif de la tranche, le plan inactif est configuré pour T2, les contextes de configuration et de données de T2 sont supposés chargés dans la LCM. Suite à une préemption ou achèvement de l'exécution de T1, les deux plans commutent et la tâche T2 commence à s'exécuter dans le délai d'un cycle d'horloge. Au moment où T2 commence son exécution, la sauvegarde du contexte de la tâche T1 est initiée. Dans [Garcia et al., 2009], les auteurs font une étude de comparaison sur le coût d'une préemption et prouve que, grâce à l'architecture OLLAF, le surcoût de la préemption est 500 fois plus petit que les meilleures architectures comparées.

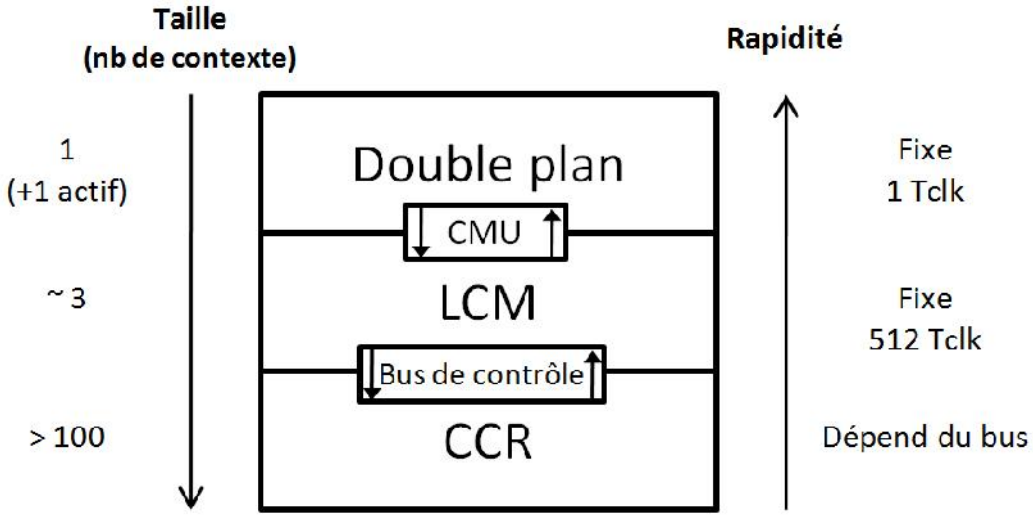

Figure 2.10. La hiérarchie mémoire des contextes

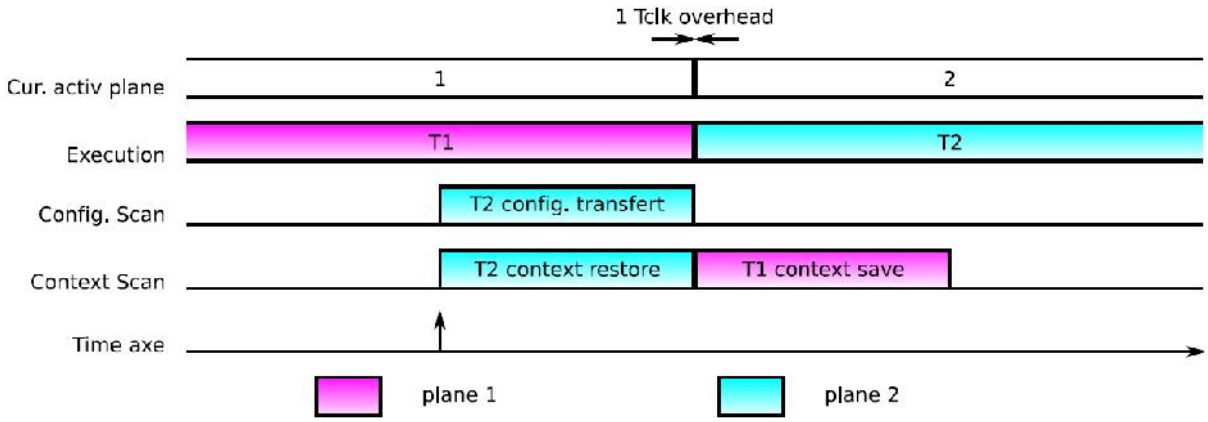

Figure 2.11. Exemple de préemption dans OLLAF

Cette amélioration est possible grâce à l'intégration de mémoires caches locales. Afin d'utiliser au mieux ces mémoires, il est nécessaire de pouvoir prédire les tâches à exécuter. En effet, dans le cas d'une exécution statique, où les données de l'application sont connues et

déterminées à l'avance, le surcoût temporel du à la reconfiguration peut être masqué grâce au préchargement des contextes des tâches. Cependant, dans le cas d'une exécution dynamique, où des variations et des perturbations se produisent, l'exécution des tâches est incertaine. Nous avons concentré nos travaux sur des applications à caractères dynamiques, nous désirons dans ce cadre introduire un service de prédiction lors de l'ordonnancement.

5. L'ordonnancement temps réel

Les applications que cible l'architecture OLLAF sont des applications temps réel. Ces applications sont à caractère réactif puisqu'elles réagissent toujours aux stimuli qu'elles reçoivent de leur environnement. Selon le respect des contraintes temporelles, on peut distinguer deux types d'applications temps réel :

- Temps réel strict ou dur : dans ce cas toutes les contraintes temporelles du système doivent impérativement être respectées. Ainsi, et quelque soit l'évènement produit, le système doit être capable de chacune garantir toujours le respect de ses contraintes temporelles. Ceci suppose qu'on puisse disposer de suffisamment d'informations sur le système pour déterminer tous les scénarios possibles. Ce qui n'est pas notre cas d'étude. Nous sommes dans ce cas en présence d'une contrainte sur la latence.

- Temps réel souple ou mou : ce type est moins exigeant en ce qui concerne le respect des contraintes temporelles. Ainsi, on trouve un certain degré de tolérance envers les fautes temporelles. La contrainte ici est liée à la cadence, c'est ce type de temps réel que nous considérons dans nos travaux.

Le problème de l'ordonnancement dans un système consiste en la définition d'un ordonnanceur, c'est-à-dire un module chargé de calculer les dates d'exécution optimales des tâches qui composent le système pour utiliser au mieux les ressources matérielles disponibles. Un ordonnanceur doit assurer le bon déroulement des traitements nécessaires tout en respectant les contraintes temporelles qui sont imposées. De nombreux travaux ont été réalisés pour résoudre le problème d'ordonnancement des tâches dans les systèmes temps réel. Dans ces systèmes la validité (aucune tâche ne manquera son échéance) et l'optimalité (l'ordonnancement proposé est le meilleur parmi les ordonnancements valides) sont deux critères importants pour décider de la qualité d'un algorithme d'ordonnancement (en pratique, on se contente souvent d'ordonnancements valides).

Pour définir un problème d'ordonnancement dans un système temps réel, il est nécessaire de connaître le modèle du système, la nature des tâches et les objectifs de l'algorithme d'ordonnancement [Benkermi 2007] :

- Le modèle du système (architecture, contraintes matérielles, mémoire, placement, communication, …)
- La nature des tâches à ordonnancer (architecture fonctionnelle de l'application, contraintes temporelles, contraintes de ressources et contraintes de précédence, priorité des tâches, préemption, …)
- Les objectifs de l'algorithme d'ordonnancement (ordonnancement optimal/faisable, hors-ligne ou en-ligne, …)

5.1. Caractéristiques générales

On peut classifier les algorithmes d'ordonnancement des systèmes temps réel, selon certains critères, dans des catégories différentes. Parmi ces critères on cite :

- Ordonnancement hors ligne : il est établit avant l'exécution réelle du système. Ainsi, l'ensemble des tâches de l'application, leurs priorités ainsi que certaines de leurs caractéristiques (e.g. temps d'exécution, l'échéance, contraintes de précédence) sont supposées connues et fixées à l'avance. Ce type d'ordonnancement est dit aussi statique : il fournit une politique d'exécution fixe tout au long de l'exécution du système. Cette caractéristique est utile pour les systèmes temps réel dur. Cependant, cet ordonnancement exige souvent de revoir l'ordre d'exécution totale de l'application après la moindre modification des caractéristiques d'une tâche quelconque [Leung, 2004].
- Ordonnancement en ligne : les décisions de cet ordonnancement sont prises lors de l'exécution réelle et ce en fonction des paramètres disponibles à l'instant de la prise de décision. Il s'agit d'une politique flexible qui s'adapte à l'évolution du système. Le choix de l'ordre d'exécution des tâches requiert une quantité de temps qui peut être non négligeable. Cet ordonnancement est dynamique et flexible, mais manque de la prévisibilité.

Il faut bien différencier les notions statique et dynamique, qui décrivent des propriétés de l'algorithme d'ordonnancement, et les notions en ligne et hors ligne qui précisent le moment où l'algorithme d'ordonnancement est utilisé. Ainsi, pour développer un système temps réel, un ordonnancement (analyse) hors ligne doit toujours être établit sans se soucier que l'algorithme final soit statique ou dynamique [Stankovic et al., 1998].

- Ordonnancement préemptif : il autorise l'opération de réquisition d'une tâche au profit d'une autre tâche plus prioritaire.

- Ordonnancement non-préemptif ou coopératif : il n'exécute une tâche qu'après le blocage ou l'achèvement de tâches. Il est plus simple à implémenter que l'ordonnancement préemptif.

5.2. Ordonnancement temps réel

Dans un système temps réel, le but principal de l'ordonnancement est de permettre le respect des contraintes temporelles associées à l'application et aux tâches. Chaque tâche possède un délai critique (date d'échéance ou *deadline*) qui est le temps maximal dont elle dispose pour s'exécuter depuis sa date de réveil.

Généralement, les systèmes embarqués sont des systèmes temps réel dont les applications nécessitent le respect des contraintes temporelles. Ceci nécessite des tests d'acceptabilité qui prennent en compte les paramètres temporels des tâches et notamment les temps d'exécutions des tâches. Il faut pouvoir connaître ces temps d'exécutions et surtout pouvoir les borner. Les tests d'acceptabilité sont des conditions nécessaires et/ou suffisantes : par exemple, pour l'algorithme Rate Monotonic ou RM (voir ci-dessous), le test d'acceptabilité pour un système composé de n tâches, qui peut être réalisé hors ligne, nous est donné par la formule suivante :

$$\sum_{i=1}^{n} C_i/T_i \leq n(\sqrt[n]{2} - 1)$$

où, pour chaque tâche i de [1,…, n], Ci est le temps d'exécution de la tâche et Ti est sa période.

L'ordonnanceur désigne un composant du noyau du système d'exploitation qui choisit l'ordre d'exécution des processus par les ressources. Un des rôles du système d'exploitation, et

plus précisément de l'ordonnanceur du noyau, est de permettre à tous ces processus de s'exécuter et d'utiliser les ressources de manière optimale. Pour arriver à donner l'illusion que plusieurs tâches sont traitées simultanément, l'ordonnanceur du noyau du système s'appuie sur les notions de commutation de contexte et d'ordonnancement. Dans [Buttazzo, 2004] on trouve une description des algorithmes d'ordonnancement les plus connus, et dont on cite :

- Rate Monotonic (R.M.) [Sha et al., 1994] : est un algorithme d'ordonnancement temps réel hors ligne à priorité fixe. Il attribue la priorité la plus forte à la tâche qui possède la plus petite période. Il n'est généralement utilisé que pour ordonnancer des tâches vérifiant ces propriétés.

- Deadline Monotonic (D.M. ou Inverse Deadline) [Audsley et al., 1991] : il s'agit d'un algorithme hors ligne préemptif à priorité constante (fonction du délai critique de la tâche). On suppose que l'échéance relative d'une tâche est au plus égale à sa période. La priorité la plus forte est affectée à la tâche ayant la plus courte échéance. L'algorithme DM est optimal dans la classe des algorithmes à priorités fixes et dans un contexte de tâches périodiques indépendantes à départs simultanés et à échéances inférieures aux périodes.

RM et DM sont des stratégies d'attribution de priorité dans une politique d'ordonnancement à priorités fixes (Fixed-Priority Preemptive, FPP). Cette politique nécessite que l'on fixe hors-ligne les priorités relatives des tâches.

- Earliest Deadline First (E.D.F.) [Balarin et al., 1998] [Anderson et al., 2005] est un algorithme d'ordonnancement préemptif à priorité dynamique utilisé dans les systèmes temps réel. Il attribue une priorité à chaque requête en fonction de l'échéance de cette dernière. Les tâches sont classées selon leurs échéances : une tâche est d'autant plus prioritaire que sa date d'échéance est plus proche. De cette manière, au plus vite le travail doit être réalisé, au plus il a de chances d'être exécuté. Cet algorithme est optimal pour tous types de système de tâches, cependant, il est assez difficile à mettre en œuvre et est donc peu utilisé. L'algorithme Earliest Deadline First est optimal pour une application temps réel composée de tâches indépendantes à échéances inférieures ou égales aux périodes.

- Least Laxity First (L.L.F.) est assez similaire à l'algorithme EDF. Il se base sur la laxité des tâches (c'est l'écart maximal entre la date d'activation de la tâche et sa date

de démarrage de telle sorte que l'échéance soit respectée). L'algorithme consiste à sélectionner la tâche qui a la plus petite laxité dans le temps. Les priorités sont dynamiquement attribuées en fonction des laxités, au fil du temps. Il est optimal sur un mono-processeur et meilleur qu'EDF en multiprocesseur. Mais il est difficile à implanter car il nécessite de maintenir à jour le temps de calcul déjà consommé par chaque tâche. En effet, pour LLF, il est nécessaire de mettre à jour fréquemment les priorités des tâches alors que pour EDF ce n'est fait qu'à l'activation de la tâche. L'algorithme Least Laxity First est optimal pour une application temps réel composée de tâches indépendantes à échéances inférieures ou égales aux périodes.

5.3. Ordonnancement pour les architectures reconfigurables

Plusieurs travaux ont été orientés vers l'ordonnancement d'applications statiques avec ou non un ordonnancement en ligne. Or, de part leurs propriétés, les DRA ont une réelle adéquation avec des applications ayant un comportement dynamique lors de leur exécution. Il est alors important de concevoir des ordonnanceurs adaptés à ce genre d'application et prenant en compte les propriétés des DRA.

Dans [Danne et Platzner, 2005], les auteurs proposent deux algorithmes d'ordonnancement préemptifs pour le traitement des tâches périodiques. Le premier, appelé Earliest Deadline First - Next Fit (EDF-NF), est une adaptation de la technique EDF pour les FPGAs. L'algorithme montre une bonne performance d'ordonnancement (faisable pour un ensemble de tâche utilisant jusqu'à 85% du système). Cependant, cette approche est commode seulement pour un petit nombre de tâches car il n'existe pas de test d'ordonnançabilité efficace. En pratique, un ensemble de tâche en cours d'exécution peut être préempté par l'exécution d'une nouvelle tâche plus prioritaire. Un inconvénient majeur de cet algorithme est que, pour chaque changement dans la liste des tâches en cours exécution, le circuit FPGA est totalement reconfiguré. Selon le nombre des préemptions et des changements des priorités des tâches, le nombre de reconfiguration peut devenir grand. L'autre algorithme présenté dans [Danne et Platzner, 2005] est appelé Merge Server Distributed Load (MSDL). Dans cet algorithme, on trouve la définition de serveurs qui sont des tâches périodiques. Ces serveurs allouent spatio-temporellement le FPGA à des tâches regroupées en leurs seins. Les différents serveurs sont ordonnancés séquentiellement sur le FPGA. Les simulations montrent que cette méthode a l'avantage d'être applicable à un grand nombre de tâches mais avec un faible taux

d'utilisation du circuit reconfigurable autour de 40%. De plus, le concept de serveur limite le nombre de configurations du FPGA et par conséquent les besoins en mémoire (pour les sauvegardes des contextes de configurations).

Resano et al. ont analysé le surcoût temporel du processus de reconfiguration et son impact sur le temps d'exécution de l'application [Resano et al., 2004]. Ils ont démontré que ce surcoût peut être réduit grâce à un ordonnancement approprié [Resano et al., 2005a], [Resano et al., 2005b]. Ils ont proposé dans [Resano et al., 2008] une solution hybride mélangeant ordonnancement hors-ligne et ordonnancement en ligne. Les résultats obtenus ont montrés la diminution du surcoût avec moins de ressources utilisées qu'en ordonnancement en ligne.

Une autre méthode pour réduire ce coût est basée sur la mise en place d'une gestion de la reconfiguration dynamique. Plusieurs services de gestion de reconfiguration permettent d'en gérer les différents aspects. On peut classifier ces services selon trois catégories :

- Optimisation temporelle, par exemple le préchargement des tâches.
- Optimisation spatiale, par exemple la gestion de la fragmentation.
- Optimisation fonctionnelle, par exemple la gestion de la préemption.

Toutefois, l'intégration de l'ensemble de ces services dans un même gestionnaire de reconfiguration engendre une grande complexité calculatoire [Darouich, 2008].

Dans un autre contexte et à une autre échelle, nous trouvons le terme « ordonnancement » utilisé dans plusieurs domaines. Dans une entreprise de production par exemple, l'ordonnancement occupe une place particulière dans la gestion informatisée des flux de production. Ces méthodes d'ordonnancement de processus dynamiques ont à priori une forte ressemblance avec l'ordonnancement sur des architectures reconfigurables dynamiquement. Le paragraphe suivant présente un survol des différentes méthodes d'ordonnancement utilisées dans les ateliers de production.

5.4. Ordonnancement d'atelier

Dans une entreprise, un ordonnancement consiste à organiser, dans le temps, le fonctionnement d'un atelier pour utiliser au mieux les ressources matérielles disponibles dans le but de produire les quantités désirées dans le temps accordé [Trung, 2005]. Généralement,

on peut classer les problèmes d'ordonnancement selon le nombre et la configuration des machines nécessaires pour réaliser chaque tâche. Ainsi on trouve [Harrath, 2003] :

- le problème d'ordonnancement des tâches sur une machine : où l'ensemble des tâches à réaliser est fait par une seule machine. Les tâches sont alors composées d'une seule opération qui nécessite la même machine.

- le problème d'ordonnancement sur plusieurs machines parallèles où on dispose d'un ensemble de machines identiques pour réaliser les travaux. Les travaux se composent d'une seule opération et chaque travail exige une seule machine. L'ordonnancement s'effectue en deux phases : la première phase consiste à affecter les travaux aux machines et la deuxième phase consiste à établir la séquence de réalisation sur chaque machine. Selon la vitesse d'exécution des machines on trouve [Michael, 2008] :
    - les ateliers à machines identiques : toute tâche peut s'exécuter sur n'importe quelle machine avec une même durée opératoire (à condition que la machine soit libre) ;
    - les ateliers à machines uniformes : chaque machine possède sa propre vitesse et ceci indépendamment de la tâche à exécuter;
    - les ateliers à machines indépendantes (ou non reliées) : la vitesse des machines dépend de la tâche à effectuer.

- le problème d'ordonnancement à *m* stations dont chacune comporte une ou plusieurs machines en parallèle. Chaque opération doit s'exécuter sur une machine différente de celles des autres opérations. En fonction du mode de passage des opérations sur les machines, on peut distinguer trois classes d'atelier [Michael, 2008] :
    - les ateliers de type flow-shop ou ateliers à cheminement unique. La particularité de ce type d'atelier est que les produits utilisent les machines dans le même ordre. On dit que le processus d'élaboration de produits est « linéaire » (chemin identique pour tous les produits fabriqués).
    - Les ateliers de type job-shop ou ateliers à cheminements multiples. Ce problème appartient à la classe des problèmes NP-complet, et il est considéré parmi les problèmes d'optimisation combinatoire les plus difficiles. Ces ateliers sont des unités manufacturières traitant une variété de produits individuels dont la production requiert divers types de machines dans des

séquences variées. Ainsi, chaque produit utilise les ressources de l'atelier dans un ordre qui lui est propre.

- Les ateliers de type open-shop ou ateliers à cheminements libres. Les produits à réaliser n'ont pas de gamme fixée. Ils doivent subir un ensemble d'opérations sur un ensemble de machines, mais dans un ordre totalement libre.

D'un autre point de vue, le problème d'ordonnancement peut être considéré comme :

- Un problème statique si les tâches à ordonnancer ainsi que l'état initial de l'atelier sont déjà connus ;

- Un problème dynamique si au début du fonctionnement on ne connaît pas toutes les tâches à réaliser.

Un ordonnancement se décompose en deux parties toujours présentes dans l'atelier, mais pouvant revêtir une importance variable [Harrath 2003] :

- l'ordonnancement prédictif qui consiste à prévoir « à priori » un certain nombre de décisions en fonction de données prévisionnelles et d'un modèle de l'atelier ;

- l'ordonnancement réactif qui consiste à adapter les décisions prévues en fonction de l'état courant du système et des déviations entre la réalité et le modèle prévu.

Pour résoudre le problème d'ordonnancement dans un atelier, il existe plusieurs méthodes que nous pouvons classer en deux catégories [Savourey, 2006] [Ourari, 2011] :

- Les méthodes exactes : elles permettent de trouver une solution optimale en faisant une exploration de tout l'espace des solutions. Ces méthodes deviennent rapidement inutilisables pour des problèmes de grande taille (comme les applications temps réel) dont le nombre de solution croît exponentiellement. Parmi ces méthodes on trouve :
  - Les méthodes classiques liées à la théorie des graphes qui supposent que les ressources sont illimitées et qu'il n y a pas de communication entre les opérations.
  - Les méthodes de programmation mathématique utilisées pour les opérations qui n'ont pas de dépendances entre elles.
  - Les méthodes de séparation/évaluation nommées Branch&Bound qui sont des méthodes arborescentes consistant à placer progressivement les opérations sur

les ressources en explorant progressivement l'arbre de recherche décrivant toutes les solutions.

- Les méthodes approchées ou heuristiques qui offrent l'avantage de ne parcourir qu'une fraction de l'espace de recherche pour parvenir à une solution acceptable relativement au critère choisi. On perd ici l'optimalité de la solution trouvée mais au bénéfice d'un gain en temps de calcul [Merhoum et al., 2007], [Pailler, 2006]. Parmi les méthodes approchées, on distingue :
  - les méthodes gloutonnes : ce sont des méthodes qui construisent une seule solution complète. A chaque étape de la méthode on complète une solution partielle en cherchant à faire le choix d'implantation le plus avantageux pour une opération de calcul donnée. Le choix d'implantation effectué à une étape donnée est définitif, on s'interdit de le remettre en cause au cours des étapes ultérieures.
  - les méthodes de voisinage : elles partent d'une solution initiale complète éventuellement calculée par une méthode gloutonne et cherchent à améliorer cette solution. On distingue les méthodes de recherche locale ou non stochastiques, les méthodes de recherche globale, ou stochastiques ou encore méta-heuristiques, les méthodes tabou, recuit simulé, colonies de fourmis et les algorithmes génétiques.

5.5. Discussion

Dans ce chapitre, nous avons présenté des travaux sur les architectures reconfigurables et les OS. Bien qu'il existe des solutions d'OS pour le reconfigurable, le problème reste l'inter-imbrication de ces deux éléments. En effet, certains OS sont développés pour faciliter, aux non spécialistes, le développement et le déploiement d'applications comportant des parties matérielles mises en œuvre sur un FPGA. D'autres OS reposent sur des architectures commerciales existantes qui sont conçues et optimisées pour répondre à d'autres problématiques. Dans cette optique, OLLAF a été proposée pour répondre aux problématiques liées à la gestion des tâches, en particulier les coûts temporels de configuration et de préemption des tâches matérielles.

D'autre part, ce chapitre a mis l'accent sur quelques notions de bases relatives aux problèmes d'ordonnancement. En pratique, plus l'environnement d'application de

l'ordonnancement est dynamique et nombreuses sont les perturbations pouvant survenir lors de l'exécution. Ainsi, il est inutile de passer un temps énorme à déterminer une solution supposée optimale alors que quelques événements aléatoires peuvent causer une forte dégradation des performances.

Ce sont ces points qui nous ont motivés pour orienter nos travaux et les inscrire dans le cadre de l'exploitation des architectures reconfigurables dynamiquement avec prise en compte des perturbations d'exécution. Ainsi, nous nous intéressons, dans le chapitre III, à proposer un nouveau modèle de représentation des applications dynamiques que cible l'architecture OLLAF. Pour pallier au problème d'ordonnancement dans un contexte perturbé, nous avons proposé, dans le chapitre IV, un ordonnancement prédictif-réactif basé sur les approches de résolution des problèmes d'ordonnancement sous incertitudes.

6. Conclusion

Les architectures reconfigurables dynamiquement ont vécu plusieurs améliorations dans le but de supporter l'évolution des applications de plus en plus complexes. Seulement, ces architectures présentent généralement des temps de configuration et des coûts de préemption limitant fortement la dynamicité du système, rendant difficile un ordonnancement efficace. Dans ce cadre, l'architecture OLLAF a été conçue pour supporter efficacement la gestion dynamique des tâches matérielles. L'intérêt de l'architecture OLLAF est que, couplée à un OS, elle permet de former une plate-forme complète capable de gérer et d'abstraire une grande partie de la complexité liée à la mise en œuvre matérielle d'applications fortement dynamiques.

Afin de fournir une politique d'ordonnancement adéquate, des méthodes de modélisation à haut niveau d'abstraction s'avèrent utiles pour rendre le processus de développement fiable. Dans la section suivante, nous nous intéressons à la modélisation des applications dynamiques qui seront implémentées. L'objectif est d'étudier les techniques de modélisation qui permettent de tenir compte des caractéristiques très dynamiques des tâches (variation des temps d'exécution, variation de nombre des tâches dans une application...) et aussi du caractère dynamique de notre architecture (variation du nombre des ressources, reconfiguration dynamique des tâches, placement en ligne, ...).

Chapitre III. Modélisation haut niveau pour OLLAF

1. Introduction

La conception des systèmes embarqués (SoC) exigent des modèles permettant leur spécification. Les concepteurs doivent utiliser un, ou des, modèle(s) pour capturer les différentes configurations possibles et les stratégies de contrôle du système. Cela permet aux différentes décisions de conception d'être évaluées et aux alternatives de conception d'être explorées et étudiées. En général, la modélisation des systèmes est élaborée soit à l'aide d'un modèle trop simpliste et statique autorisant une analyse du comportement aisée du système mais éloignée de son comportement réel, soit par un modèle plus proche du système réel, mais dont l'étude est trop complexe [Andersson, 2005]. Nous nous intéressons en particulier aux RSoC. Ces systèmes sont caractérisés par une forte interaction avec leurs environnements et par une capacité de prise en compte des caractéristiques dynamiques d'une application.

Dans ce chapitre, nous présentons une étude sur les techniques de modélisations existantes. Nous présentons, à la suite de cette étude, une nouvelle méthode de modélisation des applications dynamiques qui seront exécutées sur des architectures reconfigurables dynamiquement telle que l'architecture OLLAF.

2. Approche AAA pour OLLAF

L'Adéquation Algorithme Architecture (AAA) consiste à étudier en même temps les aspects algorithmiques et architecturaux en prenant en compte leurs interactions, en vue d'effectuer une implantation optimisée des algorithmes sur des plateformes matérielles bien déterminées. Ceci en vue de réaliser les compromis nécessaires entre les différentes contraintes à respecter : contraintes de temps, de surface, de consommation, ... Ces contraintes doivent être respectées tout au long du cycle de développement. Pour faciliter le traitement et augmenter l'expressivité de certaines caractéristiques, on a recours à des représentations plus au moins abstraites des parties algorithmiques et architecturales. D'un côté, l'application peut être vue comme un ensemble de tâches concurrentes qui communiquent entre elles via l'envoi et la réception de données. De l'autre côté, l'architecture cible représente les composants matériels tels que les ressources de calcul, les mémoires, et l'interconnexion à travers laquelle communiquent les différents composants.

Nos travaux s'appuient sur une démarche d'adéquation algorithme architecture et visent à déterminer une mise en œuvre optimisée d'une application ayant des caractéristiques de dynamicité forte sur l'architecture OLLAF. Pour cela, une approche d'ordonnancement basée sur une modélisation haut niveau est proposée. Outre la caractérisation de l'application, notre approche permet de prendre en compte l'interaction du système avec son environnement d'exécution. Cette approche adopte le paradigme de modélisation en « Y » [Keutzer et al., 2000], [Kienhuis, 1997] qui consiste en la modélisation séparément de l'application et de l'architecture puis en une phase d'association de deux modèles. L'approche en « Y » est définie dans un cadre statique, les choix effectués le sont une fois pour toute, si des modifications sont à apporter il faut relancer le flot. Nous proposons ici d'ajouter un comportement dynamique à la phase d'association, qui pourra alors évoluer dans le temps en analysant l'historique de l'exécution d'une application dynamique sur l'architecture OLLAF. Le flot de cette approche proposée est donné par la figure 3.1. Il prend en entrée, d'une part la modélisation graphique d'une application, basée sur une représentation originale que nous présentons dans la partie 3, et d'autre part une description d'une architecture reconfigurable dynamiquement. Dans notre cas, nous utilisons la spécification de l'architecture OLLAF.

A l'aide de ces descriptions, l'ordonnanceur fait un ordonnancement dynamique basé sur un processus de prédiction. La partie encadrée en traits pointillés, visibles dans la figure 3.1, présentent le processus de prédiction qui prend les décisions en ligne basées sur les observations précédentes lors de l'exécution de l'application et en prenant en compte les contraintes de l'application. Avec notre modèle, l'ordonnanceur est en mesure de distinguer deux parties de l'application : une formée par des tâches statiques et autre contenant des tâches dynamiques. Le principe de notre approche est de réaliser un ordonnancement initial, où toutes les fonctionnalités dynamiques ne sont pas pris en compte (tous les paramètres dynamiques sont égaux à zéro). Par la suite à chaque exécution les paramètres dynamiques sont mises à jour. Sur la base de ces nouveaux paramètres, un ré-ordonnancement se fait en tenant compte de leur évolution.

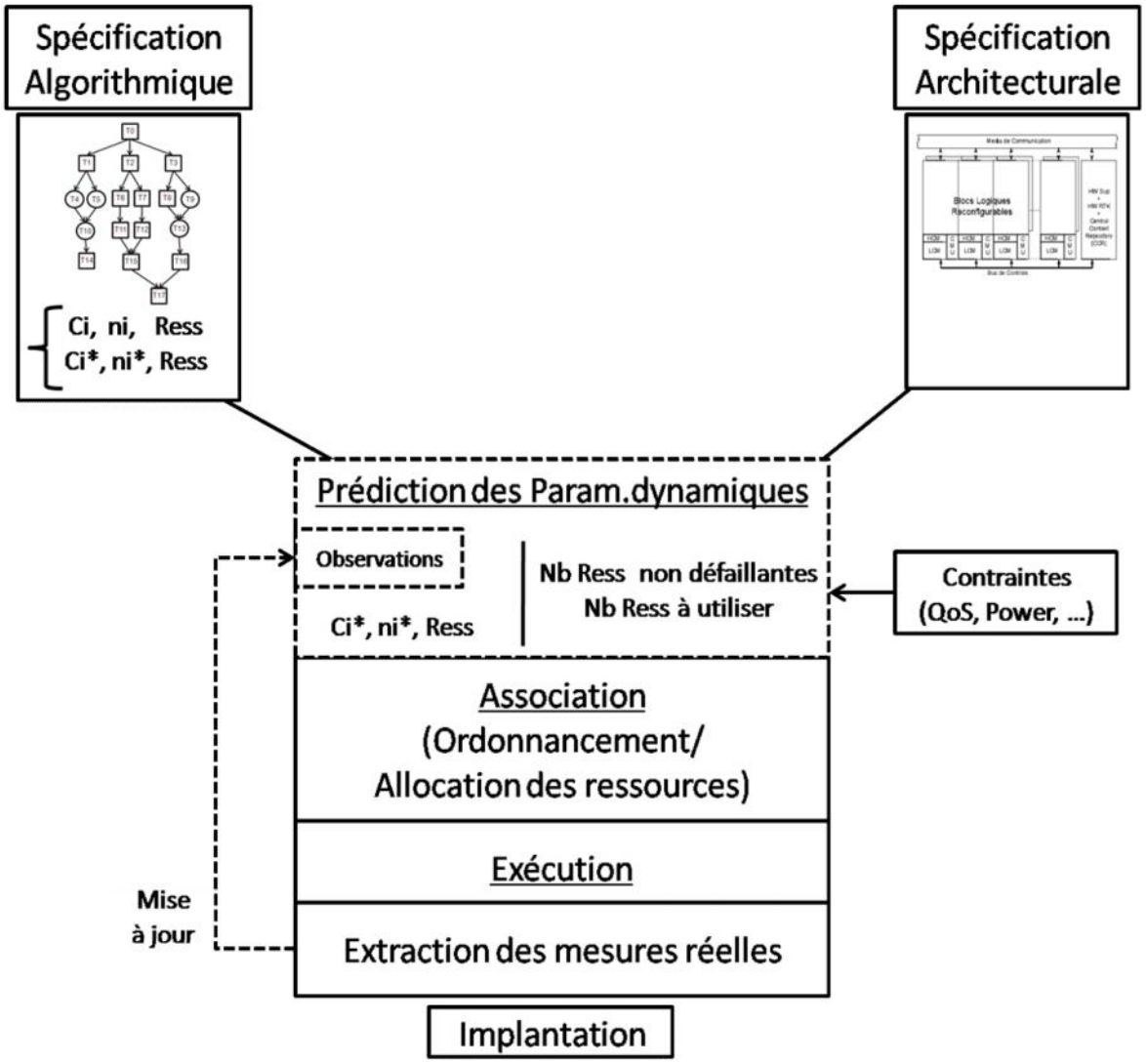

Figure 3.1. Flot « Y » d'implantation d'une application sur l'architecture OLLAF

3. Approches de modélisation

La modélisation d'un système est une représentation plus ou moins précise des éléments formants ce système. Elle peut être exprimée par des formules mathématiques, des symboles, des mots, mais essentiellement, c'est une description d'entités munies de relations entre elles [Piétrac, 1999]. Une modélisation est conditionnée par la structure du système et les règles qui relient ses éléments et ne doit en aucun cas briser les contraintes définies par le système lui-même. Elle permet une analyse du système selon un ou plusieurs critères, qui peuvent être des propriétés fonctionnelles ou non-fonctionnelles. De nombreux modèles ont été proposés dans la littérature pour représenter des systèmes sur puce. Ils couvrent un large éventail de caractéristiques et de domaines d'application. Il y a une quantité importante de résultats

théoriques et beaucoup de ces modèles ont été appliqués dans des contextes réalistes. Toutefois, des approches ciblant particulièrement les systèmes sur puce dynamiquement reconfigurables et qui prennent en compte les propriétés dynamiques de l'application, c'est à dire la manière dont un système se comporte et change d'état au fil du temps [Fishwick, 2007], ont jusqu'à présent été rares, voire inexistantes.

3.1. Techniques de modélisation existantes

Le choix de la façon de représenter un système dépend du choix des paramètres et caractéristiques que le concepteur cherche à mettre en valeur. Un modèle adopté pour un système ou un domaine d'application peut ne pas être pertinent pour un autre domaine.

Il existe différents types de modèles comme le montre la figure 3.2. Certains sont textuels, voir la figure 3.2c, et basés sur des symboles, tandis que d'autres sont associés à des graphes. On trouve des modèles qui représentent le système sous une forme statique et d'autres sous une forme dynamique. Par exemple, dans l'exemple (a) de la figure 3.2, le jeton peut circuler tout au long du réseau représentant le déroulement et le comportement de l'application durant son exécution, c'est une forme dynamique. Alors, que les exemples (c) et (d) de la figure 3.2 ne représentent que la fonction de transfert liée à l'application sans indication quand au déroulement de son exécution, c'est une forme statique.

Dans le domaine des systèmes électroniques, un grand nombre de langages et techniques de modélisation ont été proposées tels que des machines à états finis, des graphes de flot de données, les processus de communication à base de file d'attente et les réseaux de Petri [Edwards et al., 1997], [Lavagno et al., 1999], [Jantsch, 2003]. Certaines de ces techniques ont été étendues pour les adapter à des besoins spécifiques.

Dans ce qui suit nous allons présenter les principaux modèles existants dans la littérature.

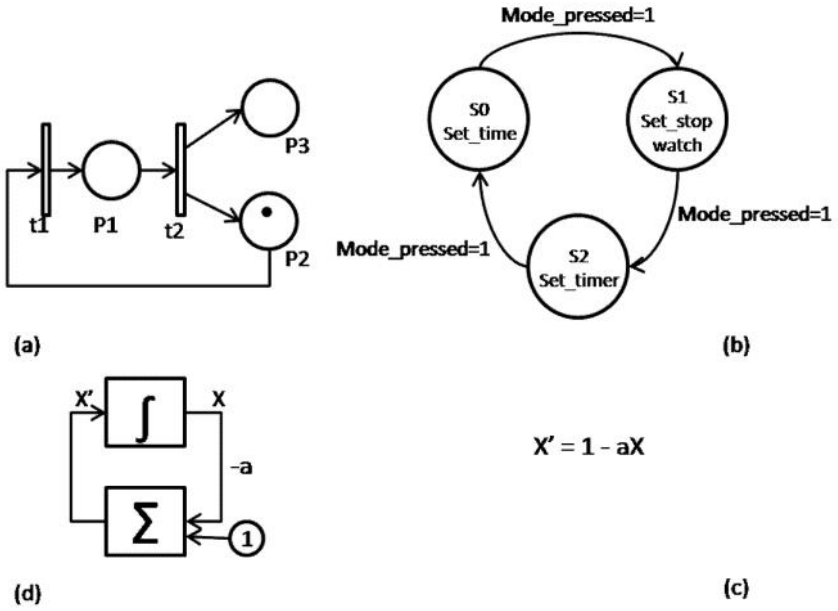

Figure 3.2. Différents types de modélisation :
(a), (b), (d) utilisant des graphes (c) type textuel

a) Machine à états finis ou automate fini (FSM)

Les machines à états finis sont les modèles les plus connus pour la description de la partie contrôle d'un système. Le modèle consiste en un ensemble d'états Q, un ensemble d'entrées I, un ensemble des sorties O, une fonction I*Q→O qui définit les sorties et une fonction de transition I*Q→Q qui définit l'évolution de la machine à travers les successions des états. On retrouve ce modèle dans la modélisation de processus, le contrôle, les protocoles de communication, la vérification de programmes, la théorie de la calculabilité, dans l'étude des langages formels et en compilation. Ils sont utilisés dans la recherche des motifs dans un texte par exemple. Un des inconvénients de ce modèle est l'accroissement du nombre des états variant exponentiellement en fonction de la complexité du système. Ceci rend le modèle difficile à analyser [Cortes, 2005]. Pour les systèmes dynamiques, cette représentation n'est pas appropriée parce que la seule façon de modéliser ce type de systèmes est de créer tous les « Etats » qui résulteraient de l'exécution dans le pire cas. Le nombre d'états risque d'être excessif et sans commune mesure avec le comportement réel de l'application.

b) Diagramme de flots de données, ou DFD

Un graphe de flot de données est un ensemble de nœuds reliés par des arcs orientés représentant le flux de données. Dans ce modèle les nœuds décrivent le traitement et les arcs représentent l'ordre partiel suivi par les données. Les calculs sont exécutés seulement lorsque

les opérandes exigés sont disponibles, c'est à dire que tous les nœuds prédécesseurs ont produit leurs résultats. Les arcs permettent de spécifier des dépendances entre les traitements. Dans ce modèle classique l'unité de contrôle n'est pas représentée [Cortes et al., 1999]. Il ne fournit aucune information sur l'ordre de commande des processus. Il est donc inapproprié pour la modélisation des applications dynamiques.

c) Réseau de Petri (Rdp)

Le réseau de Petri (RdP ou en anglais Petri Net) est un outil de modélisation qui combine une théorie mathématique bien définie avec une représentation graphique du comportement dynamique des systèmes [Fishwick, 2007]. Le réseau de Petri est composé de cinq éléments de base (P, T, F, W, M_0) [Peterson, 1981] où P est un ensemble fini de places qui représentent l'état du système avant ou après l'exécution d'une transition. T est un ensemble fini de transitions qui représentent les tâches. F est un ensemble d'arcs. W: F ➔ {1, 2, 3, ...} est appelé ensemble d'arcs primaires qui assigne à chaque arc un entier positif non nul qui indique combien de jetons sont consommés depuis une place vers une transition, ou sinon, combien de jeton sont produits par une transition et qui arrivent pour chaque place. M_0 est le marquage initial. Un RdP est défini de façon statique par sa structure et son état initial M_0. Sa dynamique d'évolution est donnée par la règle de tir (ou franchissement) d'une transition [Tavana, 2008], permettant aux jetons d'évoluer de marquage en marquage. L'ensemble des marquages, atteignables à partir de l'état initial M_0, permet de décrire l'ensemble des états que le système peut prendre. Si le réseau de Petri est bien établi pour la conception des systèmes statiques, il est difficilement utilisable pour modéliser les systèmes dynamiquement modifiables [Rust et al., 2004]. Ce modèle souffre du manque de spécifications telles que la notion de temps [Rammig et Rust, 2003]. Pour surmonter ces problèmes, plusieurs formalismes ont été proposés dans différents contextes. Par exemple les RdP temporisés dont le principe est d'associer une durée fixe aux transitions, on parle de transition T-temporisée, ou encore aux places du réseau, on parle de place P-temporisée. La temporisation consiste à affecter une durée, soit aux transitions pour exprimer la durée de l'évènement, soit aux places pour exprimer la durée séparant deux évènements. Les RdP stochastiques, qui sont un autre exemple, ont l'aptitude à modéliser le hasard dans une situation, et aussi tenir compte du temps. Il y a encore les RdP colorés qui permettent à l'utilisateur et au concepteur d'assister à des changements dans les places et les transitions grâce à l'application des jetons typés par des couleurs spécifiques, et le mouvement à travers le système peut être représenté à travers les

changements de couleurs [Fishwick, 2007]. Une comparaison entre les RdP et notre modèle sera présentée dans le paragraphe 4.3..

d) Réseau PERT

Le réseau PERT (« Program Evaluation and Review Technique » en anglais) est un modèle qui a été initialement introduit en 1958 par la marine américaine pour son système d'armes « Polaris ». Depuis, PERT s'est propagé rapidement dans presque toutes les industries. Les tâches sont représentées par des arcs auxquels sont associés des valeurs numériques correspondant à leurs durées d'exécution. Entre les arcs, il y a des cercles marquant les événements de début ou de fin des tâches (figure 3.3). Chaque nœud (i) contient trois caractéristiques le numéro de l'événement (i), un temps de début au plus tôt (Ri) pour lequel un événement peut être prévu, et un temps de fin au plus tard (Di) pour lequel un événement doit avoir eu lieu. Une tâche, située entre un nœud A et un nœud B, est considérée comme critique si la différence entre le dernier temps de fin de B et le temps de début au plus tôt de A est égale à la durée d'exécution de la tâche. Toutes les tâches critiques forment le chemin critique, qui est le chemin sur lequel aucune tâche ne doit être retardée.

Une caractéristique intéressante du modèle PERT est qu'il permet de prédire la durée minimale et maximale d'exécution de l'application en supposant que les temps d'exécution des tâches sont connus [Kerzner 2003]. Pour chaque événement, PERT indique la date au plus tôt pour qu'une tâche puisse commencer/terminer et la date au plus tard pour qu'une tâche puisse commencer/finir. Les dates au plus tôt et au plus tard sont considérées comme des variables aléatoires. Pour les estimer, un ordonnanceur peut se référer à la liste des occurrences d'événements qui ont été établies depuis le début de l'application. Cet ordonnanceur doit avoir à sa disposition un grand volume de données représentant l'histoire des exécutions passées à partir desquelles il peut faire des estimations fiables. Ce modèle est intéressant car il permet de décrire une durée variable d'une tâche, mais il ne considère pas d'autres paramètres dynamiques, comme par exemple le nombre variable de tâches ou de ressources.

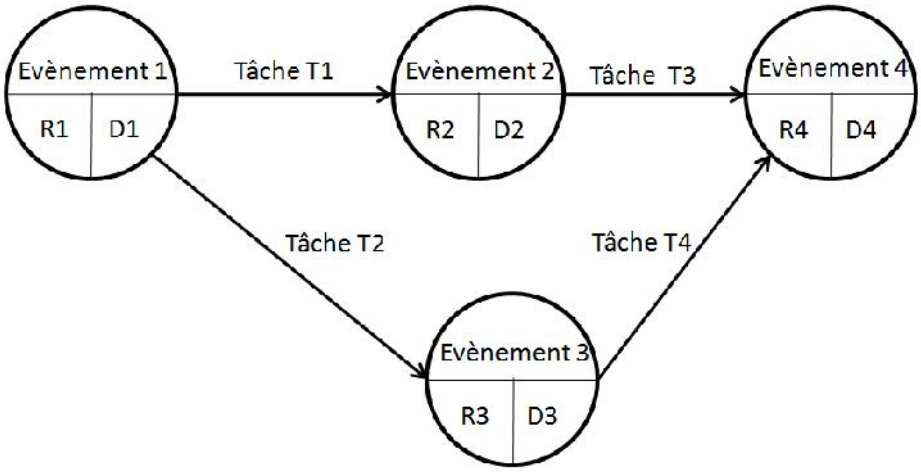

Figure 3.3. Réseau de PERT

3.2. Discussion

La plupart des modèles présentés n'offrent pas de lexique permettant de représenter des caractéristiques incertaines et imprévisibles d'une application. La création dynamique de tâches par exemple n'est pas prise en charge. Le modèle à base de machines à états ne supporte pas la création dynamique d'états, de plus, il est possible d'aboutir à une explosion combinatoire en explorant tous les chemins possibles. Les RdPs offrent un mécanisme aisé de représentation et peuvent combler les manques cités précédemment, cependant, même avec les extensions ils ne supportent pas en totalité les problèmes des exigences temporelles. Dans [Rammig et Rust, 2003], les auteurs ont proposé une extension haut-niveau du modèle de RdP [Badouel et Oliver, 1998] afin de pouvoir s'adapter aux systèmes embarqués modifiables dynamiquement. Ils ont couplé ce modèle avec des techniques de transformation de graphe et utilisé une approche push-out qui consiste à remplacer un réseau de Petri par un autre réseau de Petri après franchissement des transitions. Cette approche permet de modéliser la création dynamique de tâches, mais pas le temps d'exécution variable, ni le nombre variable de ressources de calcul.

Nous nous intéressons dans la section suivante à la modélisation des systèmes temps réel embarqués. Ces systèmes sont caractérisés par une forte interaction avec leurs environnements.

4. Modélisation haut niveau d'une application implémentée sur OLLAF

Dans cette partie une méthode pour modéliser une application présentant certaines caractéristiques incertaines est proposée et définie. Pour cela nous considérons un contexte où le temps réel est mou. Pour cette catégorie d'applications les retards d'exécution ou même les violations des délais critiques de certaines tâches influe sur la dégradation de la qualité de service sans que l'application soit défaillante. Les tâches considérées sont des tâches matérielles périodiques préemptibles qui peuvent s'exécuter sur une ou plusieurs tranches de l'architecture OLLAF.

Lors de l'exécution, un certain nombre de caractéristiques du système peuvent changer. Pour notre étude, nous considérons trois types de caractéristiques dynamiques :

- Le nombre de tâches à exécuter qui peut changer d'une itération à l'autre.
- Le temps d'exécution des tâches peut être variable.
- Le nombre de ressources nécessaires pour l'exécution des tâches est aussi variable.

De plus, le modèle considéré doit prendre en compte les trois restrictions suivantes :

- Chaque tranche d'OLLAF ne peut traiter qu'une seule tâche à la fois ;
- Chaque tâche peut être exécutée sur une ou plusieurs tranches à la fois ;
- Le nombre de tranches disponibles peut varier (suite à une panne ou destruction).

4.1. Modèle de tâche

Nous considérons des applications constituées d'un ensemble de tâches (T1, ..., Tn) suivant le modèle de Liu & Layland [Liu et al., 1973].

Dans notre modèle, une tâche $T_i = (C_i, D_i, N_i, A_i)$ est caractérisée par :

- Un délai critique D_i représentant la durée maximale pour l'exécution de la tâche T_i depuis sa dernière activation.

- Une période P_i ,
- Une durée d'exécution C_i, c'est à dire la durée que l'ordonnanceur doit consacrer à une tâche pour s'exécuter. Cette durée peut être supposée constante à chaque activation de T_i, ou variable d'une itération à l'autre,
- Un nombre N_i d'instances possibles d'exécution de la tâche,
- Un nombre A_i de ressources nécessaire à son exécution.

Nous supposons que l'inégalité $C_i \leq D_i \leq P_i$ est toujours satisfaite. La figure 3.4 illustre ce modèle de tâches. Les trois paramètres P_i, A_i et D_i sont fixés une fois pour toute pour une tâche, au contraire des paramètres C_i et N_i qui peuvent évoluer au cours de l'exécution. Une tâche dont le paramètre N_i vaut « 1 » est considérée comme permanente, ce qui signifie qu'elle est activée à chaque période. Par contre une tâche dont son paramètre N_i est différent de « 1 » ($N_i \in IN$) est une tâche dynamique, c'est à dire que le nombre d'instance exécutées de cette tâche peut changer d'une période à l'autre.

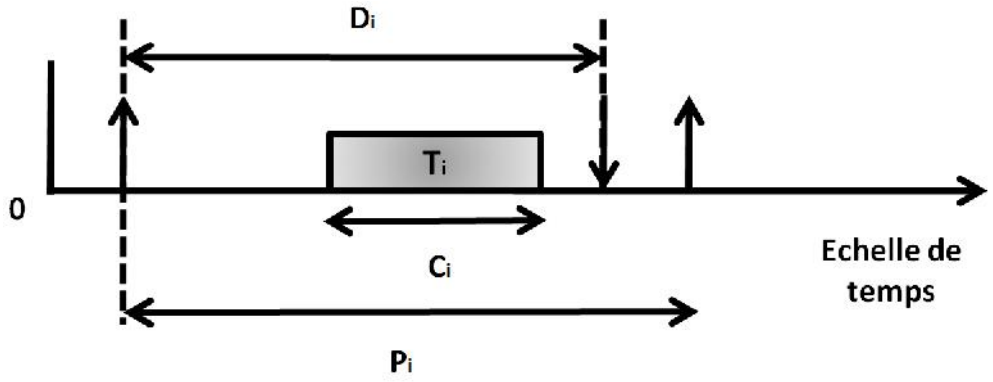

Figure 3.4. Illustration du modèle de tâches utilisé

Le modèle GMVDS (Graphical Model for Very Dynamic real-time System) que nous proposons [Ktata et al., 2010] est un triplet, GMVDS = {S, E, W}, tel que :

- S = {(T_1,H_1), (T_2,H_2), …, (T_i,H_i)} est un ensemble fini des couples (Tâche T_i, H_i), avec : i un entier positif qui rend le numéro de la tâche et H_i est un paramètre booléen qui indique si une tâche T_i est dynamique (H_i=1) ou permanente (H_i=0) ;
- E est l'ensemble des arcs du graphe ;

– W : E ➔ IN^+ est l'ensemble des poids des arcs du graphe.

4.2. Modèle de présentation visuel

Le modèle GMVDS permet de décrire deux aspects de l'application : sa partie statique, ou permanente, et sa partie dynamique, ou incertaine. Ce modèle est inspiré des modèles de type diagramme de flot de données et du réseau de PERT.

Pour représenter dans le même modèle toutes les contraintes définies dans le paragraphe précédent, le graphe est composé de deux formes de nœuds illustrés dans la figure 3.5. Le premier type de nœuds, nœud rectangulaire, se réfère à des tâches permanentes (T_j) qui sont connues à l'avance, leurs caractéristiques sont déterministes, et qui s'exécutent dans chaque période et durant tout le cycle de vie de l'application. Le second type, nœud rond, correspond aux tâches dynamiques (T_i) dont les caractéristiques sont variables et ne sont pas déterminées au début de l'exécution de l'application.

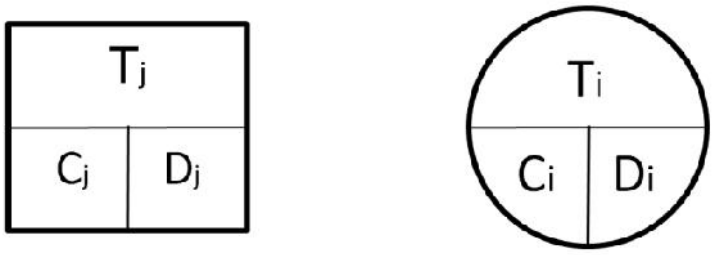

Figure 3.5. Présentation des nœuds dans le modèle GMVDS

Le premier caractère dynamique pris en compte dans ce modèle est le temps d'exécution variable. Pour représenter ce cas, nous nous sommes inspirés de PERT qui est un modèle de réseau qui prend en compte l'aspect aléatoire des temps d'exécution des activités qu'il décrit. Pour chaque tâche, PERT indique une date de début et de fin au plus tôt et plus tard. Dans le modèle que nous proposons, nous remplaçons le temps de début par le temps d'exécution qui peut être modifié en cours d'exécution, et nous remplaçons la date limite par l'échéance afin de pouvoir minimiser le temps d'exécution total de l'application, appelé *makespan*. Ainsi, chaque tâche T_k est représentée avec des caractéristiques de temps : C_k pour son temps d'exécution et D_k pour son échéance. Pour les tâches permanentes, on distingue celles avec un temps d'exécution déterministe et celles avec un temps d'exécution variable. Comme on le

voit sur la figure 3.6, le temps d'exécution variable est indiqué par une étoile : T_p et une tâche à temps d'exécution C_p déterministe, alors que T_q à un temps d'exécution C_q^* variable.

Figure 3.6. Présentation du paramètre de temps d'exécution d'une tâche permanente dans le modèle GMVDS

Pour être exécuté, les tâches matérielles ont besoin d'un nombre minimum de ressources. Cette caractéristique est notée « A_i », elle représente le pourcentage minimal de ressources nécessaire à l'exécution de la tâche. Elle est indiquée sur les étiquettes placées à côté des nœuds des tâches (figure 3.7). Nous avons représenté le nombre de ressources minimal car il se peut que la tâche demande plus que ça selon les données qu'elle reçoit. Ce cas est fréquent dans les applications multimédia, où les objets manipulés ont des caractéristiques différentes et peuvent nécessiter aucune ou plusieurs ressources. La pluralité dans notre modèle est définie par le paramètre α. Les besoins dynamiques en ressources sont représenter par le terme « $\alpha.A_i$ », avec α entier positif ou nul.

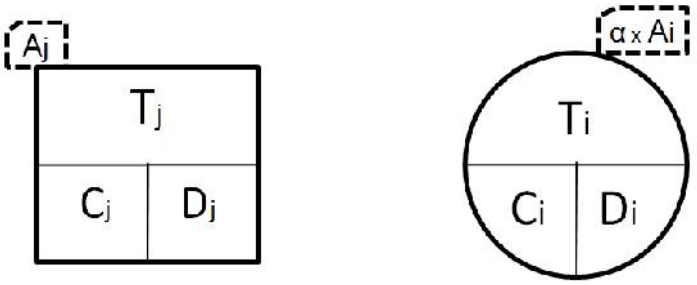

Figure 3.7. Présentation du paramètre des ressources dans le modèle GMVDS

Le paramètre « α » est aussi noté sur les arcs. Les arcs représentent les dépendances entre les tâches (figure 3.8). Pour les tâches statiques, nous représentons les arcs avec des traits pleins, tandis que les dépendances imprévisibles liées à des tâches dynamiques sont représentés en pointillés.

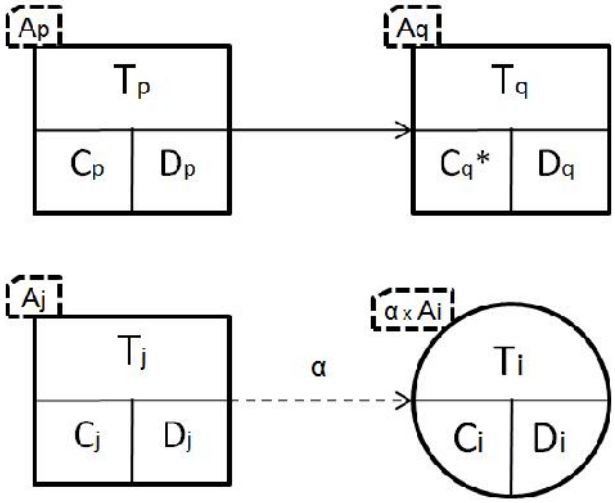

Figure 3.8. Présentation des arcs dans le modèle GMVDS

4.3. Comparaison des modèles

Soit l'exemple de la figure 3.9 où nous illustrons les différentes définitions correspondant à notre modèle. Pour cet exemple, l'ensemble {T1, T2, T3, T4, T5, T6, T7, T8} représente les tâches permanentes qui sont toujours exécutées à chaque période et {T9, T10, T11} sont des tâches dynamiques qui peuvent être exécutées dans certaines périodes, mais pas dans d'autres et dont le nombre de ressources est variable. De plus, les tâches {T1, T7, T8} ont des temps d'exécution variables, ceci est indiqué par des étoiles. La tâche T9 est une tâche dynamique représentée avec un nœud encerclé et le nombre minimum requis de ressources indiquées dans l'étiquette est multiplié par « n ». Si le nombre entier « n = 0 » alors la tâche ne se produira pas et n'aura pas besoin d'allouer des ressources. Le nombre d'instances ne sera connu que pendant l'exécution. Le nombre « n » dépend généralement de l'exécution des tâches précédentes et des données d'entrée.

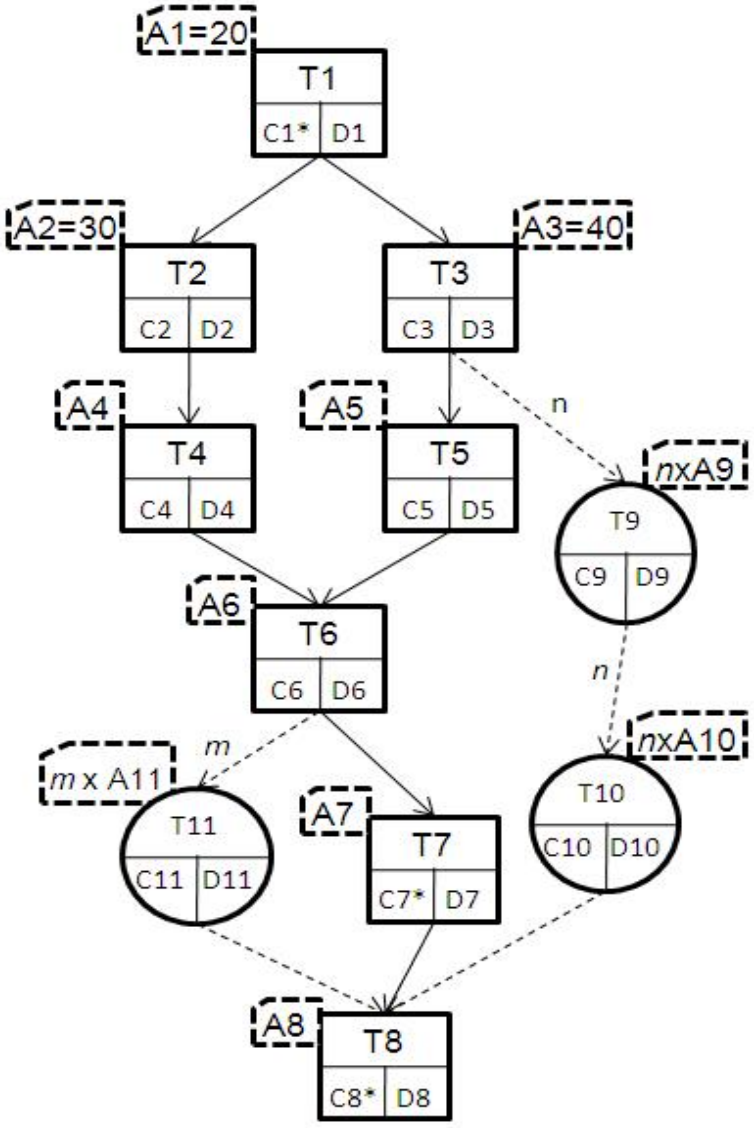

Figure 3.9. Exemple de graphe selon le modèle GMVDS proposé

Lorsque nous la comparons avec d'autres modèles, la technique proposée présente plusieurs avantages. Pour le nombre imprévu de tâches qui se produisent, un graphe de flot de données ne contient pas d'informations sur le nombre d'instances de la tâche. Pendant l'exécution de l'application, chaque tâche représentée par les nœuds est exécutée une fois à chaque itération [Sinnen, 2007]. Pour modéliser un comportement dynamique, nous avons besoin de représenter le pire cas c'est à dire les « n » nœuds de la même tâche, avec « n » maximal. La détermination de ce « n » maximal peut s'avérer complexe et irréaliste, par exemple dans le cas d'une application de suivi de cibles où « n » est le nombre de cible la maximisation peut pousser à l'extrême à associer une cible à un pixel de l'image ce qui augmente la taille du modèle sans apporter d'informations pertinente (voir figure 3.10 (b)). Dans notre modèle,

l'information sur le nombre d'instances d'une même tâche à exécuter est notée par la forme cercle de la tâche et le nombre indiqué ci-dessus de son arc. Dans le cas des RdP, voir la figure 3.10 (a), les arcs pourraient être étiqueté avec leurs poids, où un arc pondéré par « k » peut être interprété comme l'ensemble de « k » arcs en parallèles [Murata, 1989]. Mais, d'après la définition du RdP, les poids de pondération sont des entiers naturels strictement positifs, de sorte qu'il ne peut pas contenir un arc fictif avec une transition non franchissable, marquée 0, représentant une tâche qui ne peut pas être exécutée dans certaines itérations. Autrement dit, dans la figure 3.10 (a), si T9 et T10 ne sont pas exécutées, alors « n » doit être nul, ce qui est impossible à partir de la définition même du RdP. En outre, pour franchir T8, toutes les places d'entrée qui la précédent doivent avoir au moins un jeton, qui ne sera pas possible si T10 ou T11 n'a pas été exécutée, ou franchie.

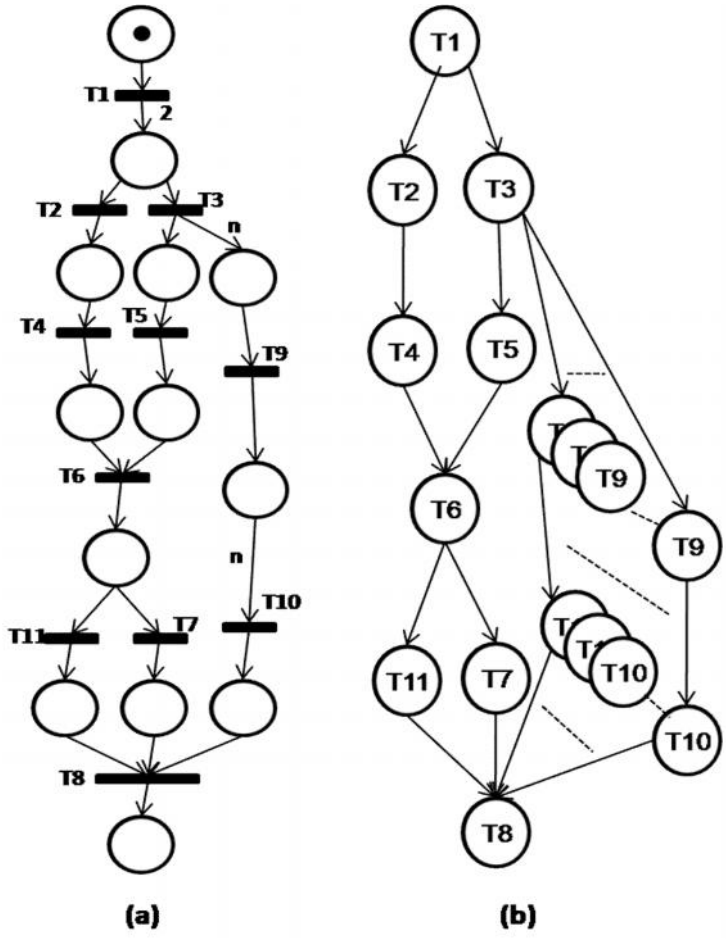

Figure 3.10. (a) Représentation en modèle de RdP de l'exemple du figure 3.9 ; (b) Représentation en modèle de DFD de l'exemple du figure 3.9

De plus, dans un réseau de Petri, pour présenter une tâche à durée d'exécution variable, il faut représenter un chemin des transitions successives par durée d'exécution ce qui complique le

modèle comme nous pouvons le constater sur la figure 3.11. L'ajout d'informations temporelles pourrait fournir une nouvelle fonctionnalité puissante pour les réseaux de Petri, mais se heurte à la philosophie d'origine de réseaux de Petri [Peterson, 1981]. Pour la représentation des ressources, le RdP modélise cette caractéristique sous forme d'une place ajoutée avec un nombre fixe de jetons. Pour commencer l'exécution, une tâche tire un jeton à partir de la place des ressources pour le restituer à la fin de son exécution. Ce modèle est difficilement utilisable pour des systèmes dans lesquels le nombre de ressources disponibles peut changer au fil de l'exécution.

Par rapport à ces modèles, le principal avantage de la méthode de modélisation que nous proposons est la possibilité de représenter plusieurs caractéristiques dynamiques des applications avec le minimum de nœuds et dans un formalisme simple. A partir de la première vue du modèle, nous pouvons faire ressortir trois caractéristiques principales de ce modèle :

- La distinction entre l'exécution statique, qui est présentée par les nœuds rectangulaires, et l'exécution dynamique c'est à dire les tâches dont l'exécution et le nombre d'occurrence est incertain, représenté par les nœuds circulaires,
- Les tâches dont le temps d'exécution est variable, représenté par une étoile,
- Le pourcentage, pour chaque tâche, des ressources nécessaires pour son exécution.

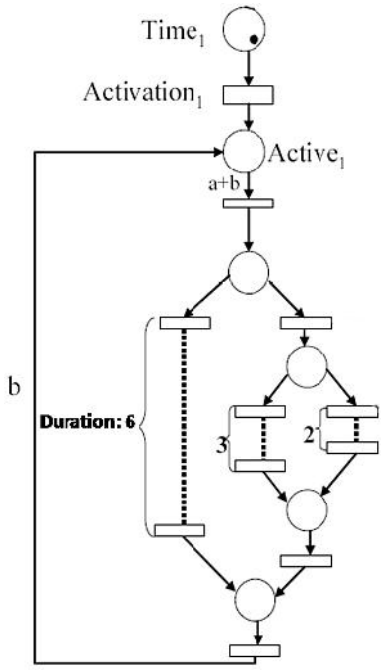

Figure 3.11. Représentation en RdP de tâche à temps d'exécution variable

5. Conclusion

La modélisation est devenue une phase importante lors de la conception des systèmes sur puce. Elle permet une abstraction haut-niveau des caractéristiques pertinentes pour pouvoir évaluer les propriétés du système. Notre modèle permet de caractériser les applications suivant leurs contraintes temporelles mais aussi par l'identification de plusieurs paramètres dynamiques : temps d'exécution, nombre de tâche, nombre de ressource. Nous utilisons ce modèle pour réaliser une adéquation en ligne d'une implantation de l'application sur l'architecture OLLAF. Cette adéquation tire profit des ressources matérielles de l'architecture. Le modèle proposé a été utilisé conjointement avec une approche d'ordonnancement efficace. Cette approche d'ordonnancement est le sujet du chapitre suivant.

Chapitre IV. Approche d'Ordonnancement Prédictif

1. Introduction

La complexité croissante des applications temps réel présente des défis de conception de systèmes embarqués importants, en grande partie en raison de leur comportement dynamique et des incertitudes qui pourraient survenir lors de l'exécution [Steiger et al., 2004]. Pour surmonter ces problèmes, les concepteurs optent pour l'utilisation des architectures reconfigurables dynamiquement. Toutefois, ce type d'architecture, très complexes, manque d'outils de gestion adaptés et efficaces [Mtibaa et al., 2007]. Cette complexité pourrait être réduite à l'aide d'outils qui pourraient intervenir soit au moment de la conception soit lors de l'exécution en fournissant par exemple un système d'exploitation abstrayant cette complexité. Dans ce chapitre une méthode d'ordonnancement sensible aux incertitudes sur les données et sur les variations de l'environnement d'exécution est présentée. Cette méthode est basée sur un ordonnanceur suffisamment souple pour pouvoir s'adapter aux éventuelles perturbations. Pour cela nous considérons une approche prédictive-réactive. Cette technique est utilisée pour faciliter l'exécution en ligne de sorte que les décisions d'ordonnancement aient une meilleure qualité et se produisent dans un temps court.

2. Caractéristiques des tâches

Notre problématique d'ordonnancement temps réel revient à définir dans quel ordre il faut exécuter les tâches matérielles sur les ressources de l'architecture OLLAF. Le but de l'ordonnancement est de prévoir avec le plus d'exactitude possible le comportement temporel du système : pré-chargement des contextes des tâches, changement des contextes, exécution, etc… .

Les caractéristiques des tâches considérées sont données par le modèle proposée dans le chapitre 3. Elles peuvent s'exécuter à des intervalles réguliers, tâches périodiques, ou de manière aléatoire, tâches apériodiques. Nous supposons que les dépendances entre les tâches sont sans perte de données, c'est-à-dire qu'une tâche ne peut entamer son exécution que lorsqu'elle dispose de toutes les données produites par ses prédécesseurs.
L'objectif est de développer une approche qui minimise le temps total d'exécution (i.e *makespan)*.

3. Approches d'ordonnancement sous incertitudes

En pratique, une méthode d'ordonnancement sous incertitude peut se découper en deux parties principales [Sotskov et al., 2010] : la phase prédictive, qui a lieu avant l'exécution de l'ordonnancement, et la phase réactive, qui a lieu pendant l'exécution.

La phase prédictive n'est pas soumise à des contraintes fortes de temps : il n'est pas nécessaire de répondre à des sollicitations en un temps imparti. Cette phase repose sur un modèle statique réalisé hors ligne. Elle peut également être appelée « phase proactive » si une méthode proactive ou proactive-réactive est utilisée.

La phase réactive a lieu pendant l'exécution de l'ordonnancement, c'est-à-dire qu'elle réagit aux événements réels qui se produisent. Durant cette phase, les décisions à prendre par le système doivent être prises rapidement, avec généralement un temps de réaction imposé. Il n'est pas envisageable d'utiliser des algorithmes très coûteux en temps de calcul dans cette phase. Comme le but de cette phase est de réagir en temps réel aux événements qui arrivent, la méthode de résolution doit posséder une connaissance exacte des événements tels qu'ils se sont passés, ils doivent être collectés hors ligne à partir d'une exécution sur le système réel, et non à partir d'un modèle comme dans la phase prédictive.

Ces deux phases sont des notions primordiales pour l'ordonnancement sous incertitudes : chaque phase possède des données différentes sur les incertitudes (du à la différence entre la connaissance exacte et celle basée sur des modèles). Nous parlons d'incertitudes pour désigner les modifications potentielles des données d'un problème d'ordonnancement qui peuvent intervenir entre le calcul d'un ordonnancement et la fin de sa mise en œuvre réelle [Esswein, 2003]. Les incertitudes correspondent donc à la différence entre les prévisions effectuées durant la phase prédictive et les données réelles obtenues après la phase réactive.

La résolution complète d'un tel problème d'ordonnancement passe par les trois phases suivantes :

- Phase 0 : C'est la phase de spécification et de modélisation du problème avec les incertitudes qu'il présente.

- Phase 1 : C'est la phase statique qui sert à calculer un ensemble de solutions, la famille des ordonnancements réalisables, par un algorithme statique hors-ligne.

- Phase 2 : C'est la phase dynamique, où une solution unique choisie parmi la famille déterminée en phase 1. Cette solution unique est calculée en ligne par un algorithme dynamique.

Les approches de résolution existantes se différencient suivant les algorithmes statique et dynamique choisis (phases 1 et 2). Ces choix dépendent bien-sûr des modèles de la phase 0. En distinguant les phases de l'ordonnancement pendant lesquelles les incertitudes sont prises en compte, une classification des méthodes d'ordonnancement sous incertitudes, qui est la classification la plus utilisée dans la littérature, permet de classer ces méthodes en trois catégories principales : les méthodes proactives, les méthodes réactives et les méthodes proactives-réactives [Davenport et Beck, 2000] [Sanlaville, 2005]. Nous présentons ici les trois catégories.

3.1. Approche proactive

Cette approche concentre son travail durant la phase prédictive. L'ordonnancement ainsi calculé sera suivi tel qu'il est durant la phase réactive. L'avantage de cette approche est qu'elle permet d'avoir le temps de prédire certaines caractéristiques pour pouvoir fournir des meilleures performances lors de l'exécution. Cependant, les calculs faits dans cette approche sont tous basés sur un modèle du système. Dans le cas où le modèle est faux, la méthode risque donc d'être rejetée. De plus, dans la phase proactive, il est impossible de prendre en compte les évènements réels qui se produisent. Ils seront donc considérés comme des incertitudes et seront modélisés. Mais si cette modélisation est fausse, alors la phase réactive sera difficile à réaliser. Cette méthode est donc adaptée aux environnements statiques et déterministes ou avec peu d'incertitudes.

3.2. Approche réactive

Les approches réactives traitent les événements réels et les prennent en considération dans les calculs de la phase réactive. Il existe deux approches différentes :

- les approches totalement réactives, qui calculent intégralement l'ordonnancement en ligne.

- les approches prédictives-réactives, qui essayent de réagir à l'évolution réelle du système en effectuant des ré-ordonnancements en ligne sur un ordonnancement statique prédéveloppé pendant la phase hors ligne.

a) L'approche totalement réactive

Elle est appelée encore approche dynamique. Le caractère dynamique vient du fait que cette approche est apte à prendre en compte les données réelles et incertaines du système. L'ordonnancement généré est établit en ligne. Il utilise généralement des techniques simples à mettre en œuvre et qui ne demandent pas beaucoup de temps de calcul. En pratique, et lors de l'ordonnancement, des heuristiques basées sur les règles de priorité (e.g., ordre d'arrivé, temps d'exécution, temps restant, etc) sont les plus appliquées [Pinot, 2008]. Le choix de la règle la plus adaptée à appliquer pour un contexte de fonctionnement donné n'est pas évident. Il est souvent la résultante de la simulation de l'application et du test de plusieurs règles de priorité. Le défaut de cette méthode est qu'elle ne prédit pas l'ordonnancement, mais le construit petit à petit. Ainsi, certaines contraintes peuvent ne pas être bien évaluées ce qui peut affecter la performance de l'ordonnancement.

Cette approche est très efficace dans un environnement peu ou pas prévisible et lorsqu'une qualité optimale (ici en terme de temps de calcul) n'est pas nécessaire.

b) Approche prédictive-réactive

De même que l'approche dynamique, cette approche ne prend en compte les incertitudes que dans la phase en ligne. La différence est que, dans ce cas, un ordonnancement est calculé lors de la phase hors ligne sans prendre en compte les incertitudes. Cet ordonnancement d'origine est utilisé au cours de l'exécution réelle du système jusqu'à ce que des incertitudes se produisent. Dans ce cas un ré-ordonnancement est effectué pour prendre en compte les nouveaux paramètres du système.

3.3. Approche proactive-réactive

La combinaison des approches déjà présentées définit l'approche proactive-réactive. Cette approche s'établit sur deux phases : la phase prédictive et la phase réactive. La phase

prédictive se caractérise par la génération de plusieurs ordonnancements de référence en se basant sur des prédictions et des optimisations, offrant ainsi plus de choix durant la phase dynamique de résolution du problème. Ces ordonnancements seront utilisés avec ou sans modifications et en temps réel durant la phase réactive selon les incertitudes rencontrées. Ainsi, cette approche profite bien des avantages des approches précédentes. L'avantage de cette méthode est que la qualité de l'ordonnancement dans le pire des cas est calculable en temps polynomial [Bidot, 2005].

3.4. Discussion

Les études sur les techniques d'ordonnancement sont très nombreuses, il est difficile d'en faire une classification exhaustive. L'approche en ligne a l'avantage d'être flexible, elle permet de prendre en compte les paramètres imprévisibles des tâches, et présente une faible complexité temporelle. Mais son inconvénient majeur est le surcoût imposé par le calcul des nouvelles priorités des tâches à ordonnancer. Quant aux techniques de l'approche hors ligne, elles sont plus optimales et n'introduisent pas un surcoût temporel lors de l'exécution puisque la solution d'ordonnancement se fait hors ligne. Cependant, ces techniques exigent la connaissance totale des paramètres de toutes les tâches présentes dans l'application. Elles sont inadaptées dans le cas de systèmes de tâches dont les dates d'arrivées ne sont pas forcément connues (tâches non périodiques). Une solution hybride peut être plus intéressante pour garantir à la fois la flexibilité du système ainsi que le respect du temps d'exécution total. Le comportement efficace d'un système temps réel face aux aléas dépend aussi du taux de bonnes prédictions de certains ordonnanceurs. Les politiques d'ordonnancement varient dans leur niveau de prévisibilité selon des facteurs comme le mécanisme déclenchant l'ordonnancement, événements ou contraintes temporelle, le temps de réaction pour prendre les décisions d'ordonnancement, la prise en compte dynamique de nouvelles tâches par l'ordonnanceur, l'existence de ressources partagées et la possibilité d'une réelle mise en œuvre de cet ordonnancement. Dans la suite de notre étude, nous avons choisi une méthode d'ordonnancement flexible et puissante permettant la mise en place efficace d'un système. Pour cela, nous avons choisit une méthode prédictive-réactive qui est un compromis intéressant permettant de faire à la fois de l'optimisation et de réagir aux incertitudes d'un système dynamique.

4. Estimation des paramètres dynamiques

En pratique, l'environnement de l'ordonnancement est dynamique et incertain. De nouvelles données peuvent apparaitre durant l'exécution de l'ordonnancement prévisionnel et le mettre en cause. Des algorithmes d'ordonnancement dynamique avec prise en compte des perturbations sont alors utilisés pour adapter progressivement la solution aux perturbations qui surgissent. La prise en compte des incertitudes lors de la construction de l'ordonnancement dépend du niveau de connaissance de l'incertitude : connue, partiellement connue ou inconnue. Dans [Billaut et al., 2005], plusieurs types de méthodes sont présentés :

- Méthode aléatoire ou stochastique où les données du problème d'ordonnancement sont associées à des variables aléatoires ou à des distributions de probabilité ;

- Méthode par intervalles où les paramètres du problème prennent leurs valeurs dans un ensemble continu de valeurs possibles. Les paramètres auxquels sont associés ce modèle sont généralement les dates de disponibilité, les dates d'échéance et les durées d'exécution ;

- Méthode par scénario où les paramètres du problème prennent leurs valeurs dans un ensemble discret de valeurs possibles. Chaque ensemble correspond à un scénario, c'est-à-dire à un problème d'ordonnancement déterministe distinct.

Généralement, les solutions d'ordonnancement lient à la fois une partie hors ligne et une partie en-ligne pour tenir compte des perturbations. Un ordonnancement prédictif-réactif doit anticiper ces perturbations et prédire leurs comportements. En pratique, la prédiction se fait sur une période appelée horizon de prédiction. Cet horizon doit être choisi de manière à ce que l'influence des données situées au-delà de cet horizon soit négligeable et que la fiabilité des données prises en compte soit assurée. Cette procédure est reprise à chaque période mettant à jour les paramètres courants du système (mesures et estimations).
La mise à jour des variables du système peut concerner l'actualisation des paramètres spécifiques à la loi prédictive, comme par exemple les dimensions des horizons de prédiction et les valeurs des termes de pondération ou bien les contraintes du problème d'optimisation. Les techniques quantitatives de prévision se divisent en deux grandes catégories : les méthodes d'extrapolation dans le temps (auto projectifs) et les méthodes explicatives [Erschler et al., 2001]. Dans les modèles explicatifs, la prévision se fonde, au moins en partie, sur des

valeurs prises par des variables autres que celle que 1'on cherche à prédire, tandis que, dans les modèles auto-projectifs ou autorégressifs, on considère que le futur se déduit tout naturellement du passé. En réalité, ces deux classes de modèles ne poursuivent pas les mêmes buts et ne s'adressent pas aux mêmes séries de données. En effet, un modèle explicatif n'est envisageable que sur un long terme et avec suffisamment de données, alors que les prédictions basées sur des modèles auto-projectifs s'utilisent sans trop de risque sur du court terme [Erschler et al., 2001]. Les techniques de prévision les plus fréquemment utilisées sont celles utilisées pour des modèles auto-projectifs. L'avantage de ces méthodes est qu'elles exigent seulement l'observation des valeurs passées. Parmi les techniques de prévisions autorégressives, nous citons [Giard, 2003] :

- La méthode de moyenne mobile qui est tout simplement une moyenne effectuée sur les *n* observations passées dans une période prédéterminée et qui se déplace au fur à mesure du temps. Selon les statistiques simples, l'exactitude de la prévision augmentera avec la longueur *n* de la série chronologique considérée, parce que les déviations aléatoires obtiennent moins de poids. L'inconvénient de cette méthode est que plus la prévision s'établit en incorporant un nombre important de valeurs passées plus le prévisionniste prendra le risque de ne pas percevoir une modification récente de la tendance.

- La méthode de lissage exponentiel qui consiste à faire une moyenne pondérée de la dernière valeur constatée et de la valeur déterminée par lissage exponentiel lors de la période précédente. Ceci évite de couper la série chronologique et nous permet de dégager une tendance. On peut dire que les méthodes de lissage consistent à réduire plus ou moins les variations en remplaçant chaque valeur par une moyenne des valeurs qui la précèdent. Pour commencer à calculer une valeur de lissage exponentiel, il faut une première valeur qui est la plupart du temps une moyenne. A partir de cette première valeur, on calcule pour chaque nouvelle période une nouvelle valeur égale à la somme pondérée de l'ancienne et de la valeur observée lors de la dernière période. Le poids des variables observées diminue exponentiellement avec la dernière variable qui a obtenu le poids le plus élevé. Par conséquent il est possible de rester à la hauteur des changements du modèle de la variable et de garder les informations qui ont été fournies par les anciennes valeurs.

- La méthode de régression qui est une équation, ou représentation graphique, permettant de chercher l'éventuelle relation fonctionnelle linéaire qui existerait entre une variable explicative (ou indépendante) x et une variable aléatoire (ou dépendante) y. Alors que le lissage exponentiel considère toutes les observations passées, la méthode de régression est appliquée à un ensemble prédéfini de données. Les inconvénients d'un tel procédé sont les même que pour le modèle de moyenne mobile. De plus, le modèle ne peut pas réagir avec souplesse aux changements des modèles de la valeur à prédire.

5. Approche d'ordonnancement proposée

En se basant sur le modèle de graphe GMVDS que nous proposons pour décrire le comportement dynamique d'une application, nous avons développé une méthode d'ordonnancement robuste qui est peu sensible aux incertitudes des données et qui permet de s'adapter à d'éventuelles perturbations. Nous nous sommes basés sur une approche proactive réactive, visible sur la figure 4.1. La technique proactive qui permet de déterminer un ordonnancement de référence calculé hors-ligne pour la partie statique est utilisée pour faciliter l'exécution de la stratégie en ligne qui est la partie réactive de notre ordonnancement. L'idée de base consiste à trier les tâches statiques à partir du modèle proposé et de les ordonnancer selon une politique de priorité dynamique, tout en respectant leurs contraintes de précédence. Les tâches sont exécutées au plus tôt. S'il existe une égalité entre certaines tâches, la tâche qui a le temps d'exécution le plus court aura la priorité la plus élevée. Lors de l'exécution, l'objectif est de générer un ordonnancement qui s'écarte au minimum de celui établi lors de la phase hors ligne, de sorte que les opérations de réparation soient très simples et peu chronophage.

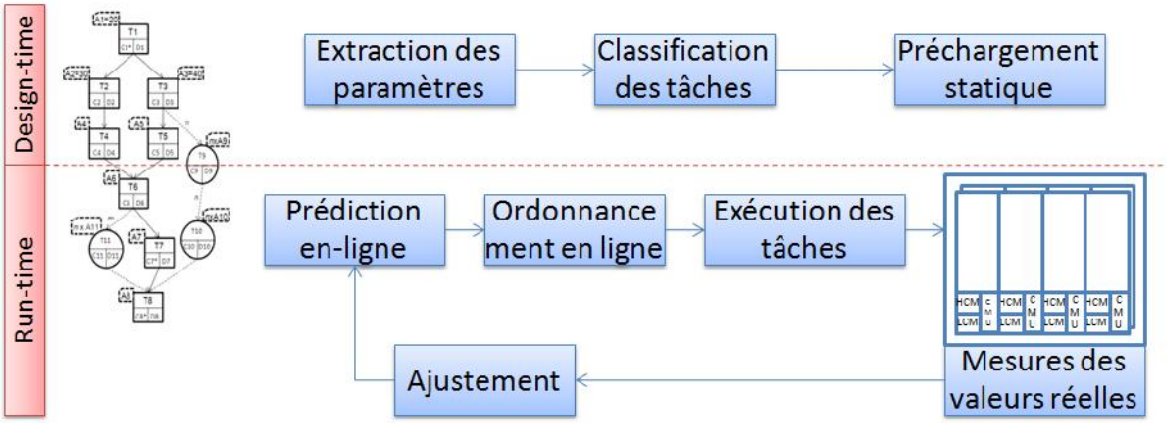

Figure 4.1. Vue globale de l'approche d'ordonnancement proposée

5.1. Phase hors ligne

Au moment de la conception (hors ligne), une analyse de l'application est réalisée. Elle vise à distinguer les caractéristiques de l'application. Chaque tâche est définie selon notre modèle par ses caractéristiques [C_i, D_i, N_i, A_i]. Pour les caractéristiques à caractère aléatoire soit C_i et N_i, elles sont exprimées par leurs valeurs maximales estimées (tel que le WCET pour C_i). Généralement les tâches sont classées en tâches périodiques/apériodiques, avec des demandes d'exécution répétitives, occasionnelles ou conditionnelles. Dans le modèle de graphe proposé, les tâches sont classées permanentes ou aléatoires. Les tâches permanentes sont toujours exécutées, elles sont périodiques et répétitives. L'exécution des tâches aléatoire est incertaine à chaque itération, elles sont apériodiques et/ou conditionnelles. A partir de l'ensemble des tâches permanentes, les tâches critiques qui appartiennent au plus long chemin, en terme de temps d'exécution, sont identifiées permettant une analyse ASAP du début jusqu'à la fin du graphe des tâches. En fait, l'exécution de ces tâches aura une grande influence sur le *makespan*. Les tâches dans le chemin critique ont plus de priorité que les autres. Nous supposons, dans un premier temps, que les tâches critiques sont toujours exécutées dans toutes les itérations de l'application. Par conséquent, leurs contextes de configuration et de données d'exécution sont chargés une fois pour toute de la mémoire centrale dans les tranches d'OLLAF, si le nombre de tranches est suffisant à l'exécution de l'application ; sinon il sera procédé à des déchargements et rechargements de contextes.

Le modèle des tâches de l'application peut contenir certaines tâches qui sont exécutées durant la presque totalité de la période du graphe. A titre d'exemple, si la période de l'application est P et il existe une tâche T_j (T_j correspond à T_1 dans la figure 4.2) telle que :

$$\left(P - C_j\right) < \min\left\{C_k, with \;\; k \neq j\right\}_{j \leq n} \qquad (1)$$

alors T_j doit conserver ses contextes de configuration et de données dans les mêmes colonnes et doit être placée de manière contiguë, pour minimiser la zone gaspillée. Ceci n'est pas une allocation restrictive, et T_j peut être préemptée tant que sa laxité le lui permet. Mais l'idée est de garder le contexte de T_j dans la mémoire cache locale LCM puisque nous sommes sûr qu'elle sera exécutée à chaque itération. Dans le cas contraire, T_j sera préemptée et migrée sur plus d'une colonne ce qui nécessite un trafic sur le bus pour aller vers la mémoire centrale pour sauvegarder et restaurer le contexte de la tâche modifiée. L'accès à la mémoire centrale est coûteux en temps et influe directement sur le temps d'exécution de l'application.

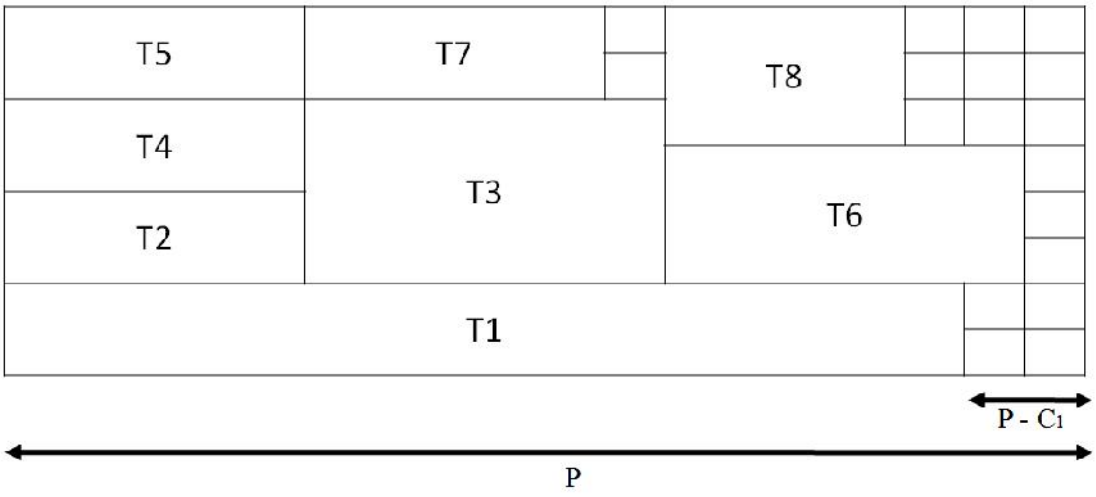

Figure 4.2. Exemple de tâche critique

5.2. Phase en ligne

Au moment de l'exécution, l'ordonnancement est effectué en ligne. Les données réelles issues de cette exécution seront collectées, les caractéristiques dynamiques de l'application ne pouvant pas être connues avant cette exécution. Ces caractéristiques doivent être estimées avant chaque itération en se basant sur les observations des exécutions antérieures faisant appel ici au mécanisme de prédiction. La prédiction n'est pas une tâche triviale : il s'agit de trouver un compromis global entre le gain et le coût (surcoût de cette prédiction en ligne). Par

conséquent, des heuristiques sont utilisées. Les techniques de prédiction que nous utilisons sont basées sur des mesures de l'historique récent telles que :

a. la valeur moyenne des trois dernières mesures réelles,

b. la moyenne pondérée des trois dernières mesures réelles,

c. le maximum des trois dernières valeurs réelles,

d. la dernière mesure majorée.

Dans la littérature, on trouve essentiellement deux approches de prédiction des temps d'exécution [Wilhelm et al., 2008] :

- Approches statiques: ces méthodes n'ont pas besoin de l'exécution de code d'application sur le matériel réel ou sur un simulateur. Elles s'appuient sur l'analyse du code pour calculer des bornes supérieures de temps d'exécution [Wilhelm et al., 2006]. Une technique d'analyse de code est généralement limitée à un type de code spécifique ou une catégorie limitée d'architectures. Ces méthodes qui analysent le code statiquement ne sont pas très applicables aux applications dynamiques. Ces méthodes d'analyse hors ligne ne prennent pas en compte les changements dans les données traitées lors de chaque activation des tâches. Les modifications en ligne de temps d'exécution ne seront pas considérées pour éviter l'échec de test d'ordonnançabilité, ou de l'utilisation excessive des ressources.

- Les méthodes basées sur des mesures : ces méthodes exécutent la tâche sur le matériel cible réel ou un émulateur matériel, pour un ensemble d'entrées données. Elles mesurent ensuite les temps d'exécution en extraient les valeurs maximales et minimales d'exécution observées à partir de leurs distributions. Ces méthodes ont des problèmes de complexité et de sécurité [Engblom et al., 2003]. Ces méthodes basées sur des mesures nécessitent de garder un historique sur des valeurs mesurées pour toutes les tâches qui forment l'application. Au fur et à mesure le temps d'exécution est mesuré et ajouté à l'ensemble des observations précédentes afin d'améliorer la précision de prédiction. Comme le nombre des observations augmente, les estimations produites par un algorithme statistique sont améliorées [Krishnaswamy et al., 2004]. Ces méthodes ne nécessitent pas une connaissance directe des caractéristiques de

l'application, des données, de la conception interne du code ou de l'architecture envisagée.

Dans les approches basées sur des mesures, généralement le problème d'estimation de temps d'exécution est considéré comme un problème de régression. Des algorithmes de régression pour calculer des estimations à partir de l'ensemble des observations précédentes sont donc utilisés. Dans la littérature, il existe deux catégories de techniques de régression : les techniques paramétriques et les techniques non paramétriques [Ghaffari, 2006]. En général, les techniques paramétriques exigent la définition d'une fonction qui décrit le temps d'exécution Y d'une tâche comme une fonction d'un paramètre X qui peut être la taille du problème, le nombre d'objets, etc…. Une technique populaire paramétrique pour résoudre ce type de problème est la méthode des moindres carrés. Cette technique exige un calcul hors ligne des coefficients de la fonction. Toutefois, pour des applications souples et dynamiques, il est difficile de faire des hypothèses sur la forme fonctionnelle. Les techniques paramétriques ne sont donc pas bien adaptées à ce problème. Les techniques non paramétriques sont une autre possibilité. Elles sont considérées comme des méthodes basées sur les données, puisque l'estimation ne dépend que de l'ensemble des observations précédentes, et non sur des hypothèses au sujet de la fonction Y(X). Une des techniques de régression non paramétriques calcule la fonction Y(X) en utilisant une variante de l'équation :

$$Y(X) = \frac{1}{n} \sum_{i=1}^{n} W_i(X).C_i \qquad (2)$$

où *Wi (X)* est une fonction de pondération. Y (X) est une moyenne pondérée des valeurs de temps d'exécution *Ci*, des *n* observations précédentes. La fonction du poids *Wi (X)* attribue généralement plus de poids aux observations proches du paramètre X, et les valeurs des poids les plus basses aux observations plus éloignées de X. Ceci est illustré dans la figure 4.3. Une technique populaire non paramétrique est l'algorithme du k-plus proche voisin ou k-NN : K Nearest Neighbors décrit dans [Iverson et al., 1996]. Pour les méthodes non paramétriques, le principal problème est qu'ils sont plus complexes à mettre en œuvre qu'une méthode paramétrique. Ils nécessitent un historique de calcul. A l'inverse des propriétés d'une tâche statique, les paramètres estimés ne dépendent pas seulement du code qui doit être exécuté par la tâche, mais aussi de l'état du système et de l'environnement au moment *t* de l'exécution. Cela signifie que les paramètres estimés sont des valeurs probabilistes qui peuvent changer au

cours de la durée de vie du système. Afin de prédire les paramètres de l'instance suivante d'une tâche T, il est nécessaire de trouver une bonne approximation à partir des exécutions les plus récentes de T.

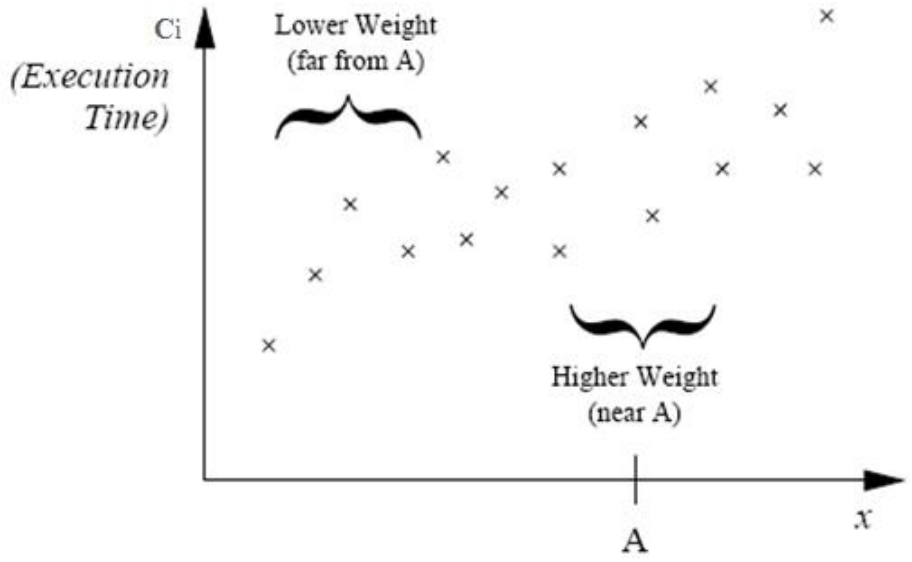

Figure 4.3. Pondération des observations précédentes.

Ainsi, et en se basant sur ces approches, l'ordonnanceur en ligne doit ordonnancer les tâches matérielles allouées sur les ressources de calculs disponibles dans l'architecture. Autrement dit, il doit charger à la fois la configuration matérielle et le contexte de donnée dans la(les) colonne(s) correspondante(s). Pour l'architecture OLLAF, ceci correspond aux blocs HCM et LCM de la figure 2.8. Ces deux niveaux de mémoire permettent la gestion du préchargement des configurations et des contextes. Le préchargement est important, il permet d'économiser le temps de configuration. Mais ceci exige de choisir la colonne la plus probable où la tâche peut être mappée. Basé sur le préchargement statique et les relations de précédence, chaque tâche dont tous les prédécesseurs ont déjà commencé leur exécution est en mesure de pré-charger sa configuration et son contexte. La priorité est attribuée tout d'abord aux tâches ayant le minimum de laxité suivant le schéma LLF, s'il y a égalité, nous passons à celles avec le temps d'exécution le plus court. Cette priorité, appelée aussi SPT « Shortest Processing Time » en anglais, a prouvé ses performances pour la réduction du temps d'exécution total [Choi et al., 2012]. Si c'est l'égalité à nouveau, nous privilégions celles nécessitant le plus de ressources. L'exécution des tâches dépend aussi de la disponibilité des ressources en ligne, de sorte que la tâche ne peut s'exécuter que si elle est prête et a suffisamment de ressources disponibles sur la zone reconfigurable.

A partir de l’exécution réelle de l’application, pour chaque caractéristique variable, et dans chaque exécution de sa tâche correspondante, les valeurs réelles sont calculées et sauvegardées pour servir lors de la prédiction des valeurs de la prochaine exécution. Cependant, prédire l’environnement d'exécution réel, y compris les données d'entrée, est évidemment limité. Par conséquent, un ajustement est nécessaire après chaque exécution afin de minimiser l'accumulation d'erreurs d'une prévision à l'autre. A partir des mesures de données en ligne, des informations sur les valeurs réelles des paramètres et la qualité du système qui en résulte, par exemple le nombre de délais critique dépassé, sont recueillies. En outre, un mécanisme de réglage est introduit dans l’application. Il est utilisé pour ajuster les valeurs des paramètres utilisés pour la prédiction.

6. Conclusion

Nous avons présenté dans ce chapitre une méthode d'ordonnancement robuste qui est peu sensible aux incertitudes de données et les variations entre la théorie et la pratique. Nous avons considéré une approche proactive réactive. La technique proactive est utilisée pour faciliter l'exécution en ligne, de sorte que les décisions d'ordonnancement soient plus rapides et plus efficaces. Pour remédier au contexte d’incertitude considérée dans nos études, nous faisons recours aux techniques de prédiction. Le chapitre suivant mettra l’accent sur l’importance de l’approche élaborée et présentera des résultats d’expérimentations.

Chapitre V. Expérimentations & Validation

1. Introduction

Tout au long de ce chapitre, nous présentons la validation de l'approche d'ordonnancement proposée au chapitre précédent à l'aide d'applications dynamiques représentatives. Nous utilisons aussi, pour la validation, des graphes GMVDS générés aléatoirement. Nos expériences ont été divisées en deux parties. Nous commençons par l'étape de modélisation, où nous nous concentrons sur la construction des graphes en se basant sur notre méthode de modélisation. La deuxième partie présente les résultats issus de l'ordonnancement proposé en utilisant les modèles des applications. Dans l'ordonnancement, nous considérons les caractéristiques de l'architecture OLLAF dont les ressources de calcul sont groupées en tranches. Chaque tranche est un tableau de 512 éléments logiques (LE) composées d'un LUT et d'une flip-flop. Une tâche est définie par autant de configurations de tranches qu'elle nécessite. Le transfert du contexte de configuration d'une tâche se fait de façon hiérarchique à travers plusieurs niveaux de mémoires. La figure 5.1 montre une vue de la hiérarchie mémoire d'OLLAF [Garcia et al., 2007]. Cette hiérarchie de mémoire est formée de :

- CCR qui est la mémoire principale,
- LCM qui constitue la cache locale de colonne,
- et ensuite le plan dual qui est le niveau supérieur et le plus rapide.

TL0, TL1 et TM représentent les trois temps de transfert entres les niveaux mémoires de l'architecture OLLAF. Le tableau 5.1 [Garcia, 2012] donne le temps de transfert du contexte d'une tranche complètement configurée en période d'horloge en supposant une fréquence de travail de 100MHz. Pour exécuter une tâche sur une des tranches d'OLLAF, il faut chercher son contexte dans la mémoire principale et le transférer dans la mémoire LCM de la tranche : c'est le temps TM. Ensuite, il faut configurer le plan caché : c'est le temps TL1. Enfin, il faut activer le plan caché pour exécuter la tâche : c'est le temps TL0. Le surcoût total en temps de transfert est donc : Tr = TM + TL1 + TL0. Cette structure complexe de transfert de configuration montre tout son intérêt si on est capable de pré-charger les tâches avant que leur exécution ne soit demandée par l'ordonnanceur. La prédiction des paramètres incertains des tâches permettra de préparer ces tâches et pré-charger leurs contextes avant que leurs exécutions effectives ne soient demandées par l'ordonnanceur principal.

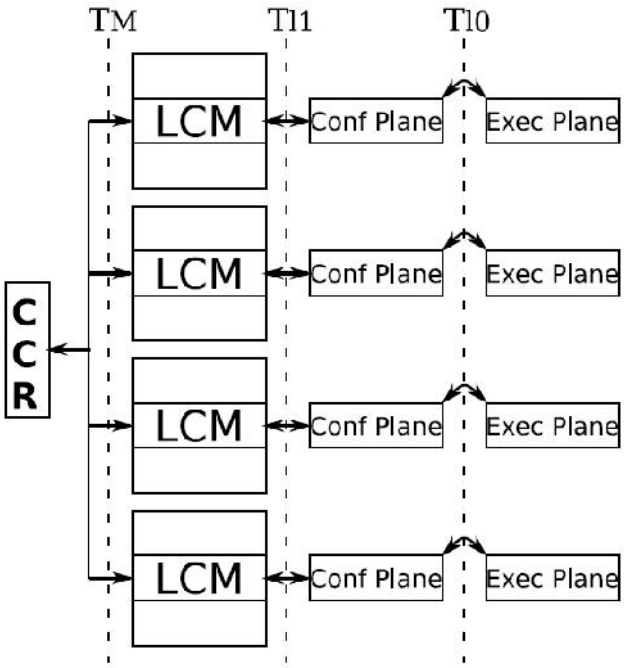

Figure 5.1. Hiérarchie mémoire de l'architecture OLLAF

Tableau 5.1. Temps de transfert pour chaque niveau de l'hiérarchie mémoire d'OLLAF

	TM	**TL1**	**TL0**
Durée de transfert (#Tclk)	1696	512	1
Temps de transfert	16,96 µs	5,12 µs	10 ns

2. Exemple de graphe de tâches générées aléatoirement

La figure 5.2 correspond à un graphe de tâches générées aléatoirement. Pour chaque tâche T_i, le temps d'exécution et le nombre d'instances ont été généré aléatoirement avec la supposition que $P_i = D_i$. Le graphe étudié regroupe des branches parallèles et est composé de dix-huit tâches périodiques pouvant être préemptées par l'ordonnanceur qui les gère. Chacune de ces tâches est caractérisée par son propre temps d'exécution, son échéance et sa priorité. Ce graphe présente les caractéristiques dynamiques suivantes : des tâches dynamiques {T4, T5, T9, T10, T13, T14, T16}, des tâches avec temps d'exécution variable {T1, T7, T8, T15}. Pour définir les paramètres incertains des tâches formant le graphe, nous avons exprimé leurs variations sous la forme de distributions probabilistes. Beaucoup de densités de probabilité sont reconnues pour représenter l'évolution de certains types des caractéristiques du monde réel [Gubner, 2006] [Fortier et al., 2003]. Pour les tâches {T1, T7, T8, T15}, les variations de

leurs temps d'exécution sont affectées à la distribution de Gumbel [Veysseyre, 2006] pour prédire le temps d'exécution. Cette distribution est démontrée dans [Edgar et Burns, 2001] comme l'évaluation la plus réaliste pour modéliser le temps d'exécution au pire cas. Pour les tâches {T4, T5, T10, T14} et {T9, T13, T16} leurs variations d'instances sont respectivement affectées à la loi de Poisson et à une distribution Gaussienne (ou loi normale). La distribution de Poisson est une fonction de densité largement utilisée pour décrire des événements rares et les phénomènes d'arrivées aléatoires [Gubner, 2006]. En outre, la distribution gaussienne est l'une des distributions de probabilité les plus importantes, et probablement la plus largement utilisée [Gubner, 2006].

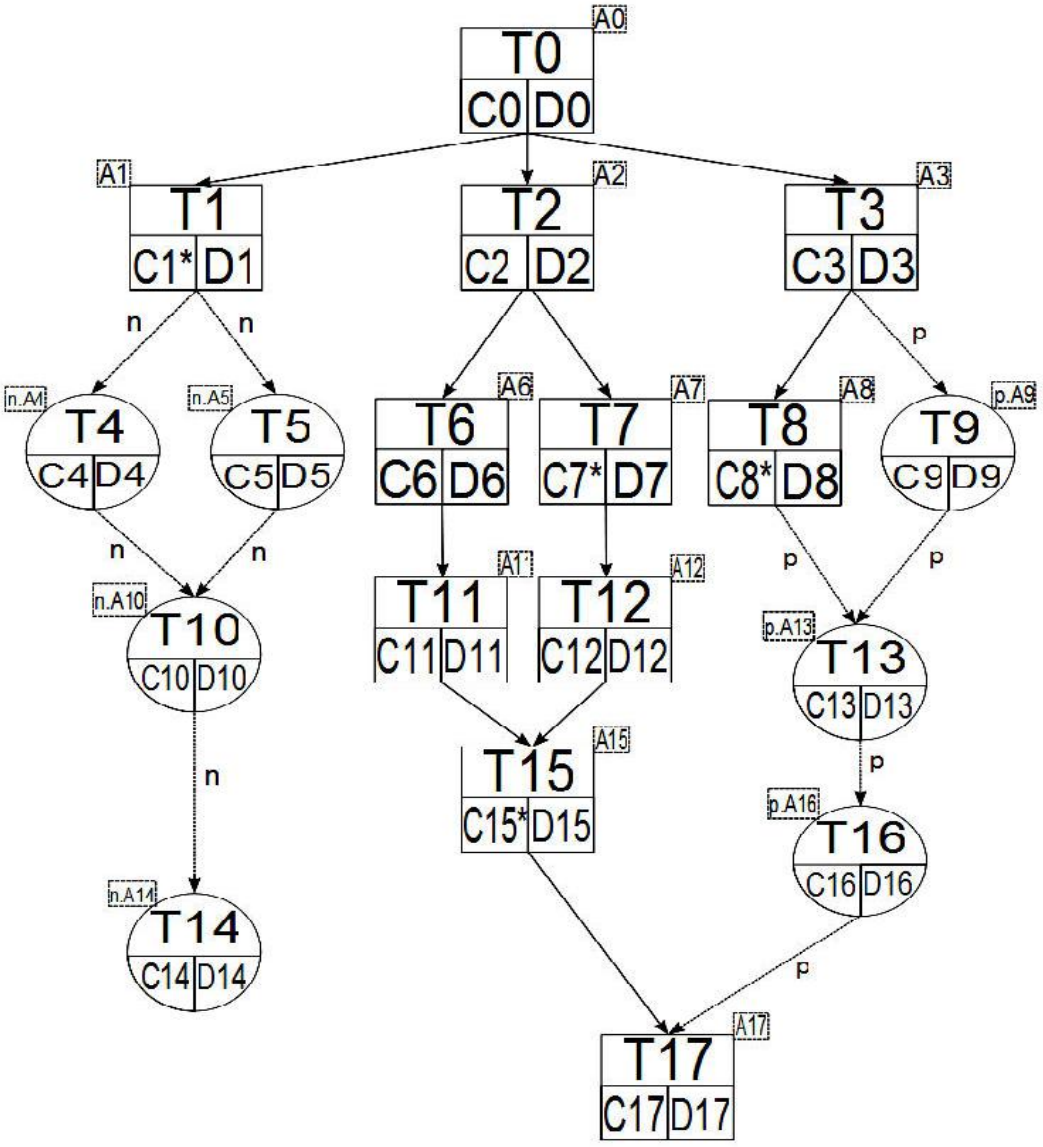

Figure 5.2. Modèle de graphe généré aléatoirement

Ce modèle est une entrée de l'ordonnanceur qui a été écrit en langage C. La figure 5.3 correspond à la sortie de l'ordonnanceur et montre le diagramme de Gantt de l'exécution des tâches du graphe pour six itérations successives (chaque itération est différenciée par une couleur différente). Le tableau 5.2 montre que notre méthode d'ordonnancement permet de réduire jusqu'à 18% le temps d'exécution total par rapport aux méthodes basées sur le WCET. En outre, les techniques de prévision de préchargement et la réutilisation conduisent à un taux de réduction important du nombre de transfert des contextes (jusqu'à 85%).

Tableau 5.2. Résultats d'exécution du graphe généré

	Première itération			Les 5 itérations suivantes			**Surcoût reconfigurations (#Tclk)**	**Cmax (ms)**
	#TM	#TL1	#TL0	#TM	#TL1	#TL0		
OLLAF sans prédiction	39	10	16	36	9	15	399675	219
OLLAF avec prédiction	3	1	8	25	19	49	57785	179

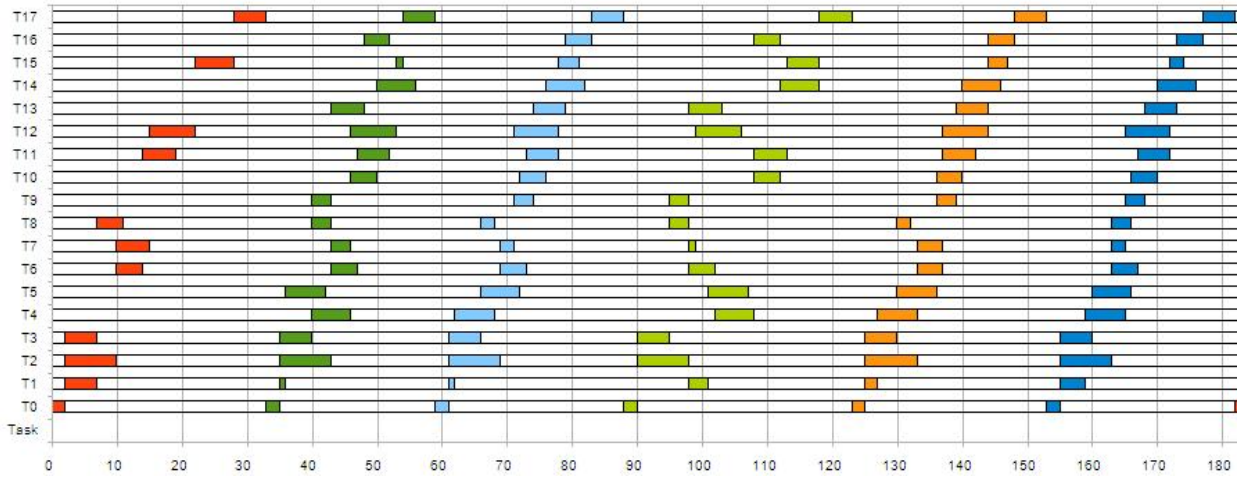

Figure 5.3. Diagramme de Gantt du graphe présenté dans la figure 5.2

3. Application de vision robotique

3.1. Présentation

Comme première application de l'approche de modélisation et d'ordonnancement décrites dans le chapitre précédent, nous avons étudié une application de traitement d'image d'un

système visuel embarqué dans un robot mobile. Dans cette application, un robot mobile doit pouvoir évoluer dans son environnement, pour cela il utilise un algorithme de détection de points d'intérêt [Verdier et al., 2008]. Les points d'intérêt correspondent à l'image filtrée par une différence de gaussiennes ou DoG. Cette application est dynamique dans le sens où le nombre de points d'intérêt dépend de l'environnement visuel du robot et n'est pas connu à priori. Il existe, de plus, trois modes de fonctionnement de l'algorithme de détection liés au comportement du robot. Chaque mode exécute des fonctions différentes avec des contraintes et des paramètres de traitement différents. Dans cet algorithme il existe jusqu'à trois échelles de représentation de l'environnement. Une échelle grossière, appelée échelle 3, une échelle un peu plus détaillée, appelée échelle 2 et une échelle très détaillée appelée échelle 1. Le principe de fonctionnement de cette application est visible sur la figure 5.4 [Lefebvre, 2012].

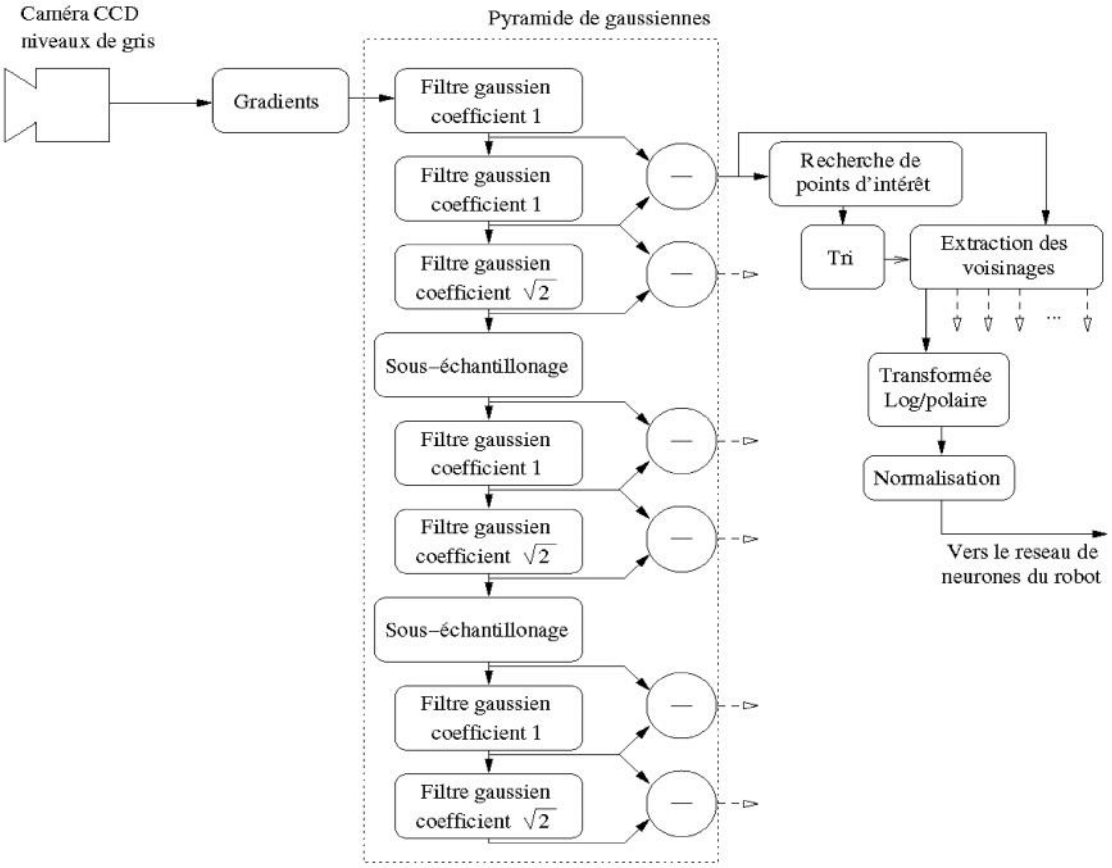

Figure 5.4. Description fonctionnelle de l'application de vision robotique

Les trois modes d'utilisation de l'algorithme sont :

a) Le mode rapide : le robot se déplace rapidement avec une description grossière de son environnement. Seules l'échelle inférieure, échelle 3, est traitée. C'est la raison pour

laquelle dans ce premier mode, le nombre maximal de points d'intérêt extraits est fixé à N = 10, et le nombre de trames traitées par seconde est de 20 images par seconde.

b) le mode intermédiaire : le robot se déplace lentement, par exemple, lorsque le passage est bloqué, l'évitement d'obstacles, le passage de porte etc... et il a besoin donc de plus de précision sur son environnement. Dans ce mode, l'échelle du milieu et l'échelle inférieure (l'échelle 2 et l'échelle 3), sont traitées. N est fixé dans ce cas à 30 et le système fonctionne à un taux de 5 images par seconde.

c) le mode lent : le robot est arrêté dans une phase de reconnaissance, suivi d'objets, ou d'exploration d'un nouveau lieu. Toutes les échelles, échelle 1, échelle 2 et échelle 3, sont entièrement traitées et des informations complètes sur l'environnement visuel sont fournies. Le nombre de processus diffère en fonction du nombre de points d'intérêt dans les trames vidéo. Pour ce mode, N = 120 et le taux du système est fixé à une image par seconde.

Les tâches de l'algorithme peuvent être divisées en trois groupes:

- Tâches de calcul intensif de flux de données qui s'exécutent en un temps constant,
- Tâches dont le nombre d'occurrence d'exécution est en corrélation avec le nombre de points d'intérêt,
- Tâches avec un temps d'exécution imprévisibles qui dépend des caractéristiques des images.

3.2. Modélisation

La figure 5.5 montre la modélisation de l'algorithme de vision robotique. On peut remarquer la présence de la branche statique, nœuds carrés, qui représentent des tâches permanentes qui seront exécutées dans tous les cas (ou modes) de l'application, c'est le cas par exemple des tâches T1, T14, T19, et les tâches aléatoires qui peuvent survenir lors de l'exécution. Le tableau 5.3 indique les fonctions des tâches formant l'application ainsi que leurs classes dans le modèle : permanentes ou aléatoires. Les tâches dont le temps d'exécution est imprévisible sont indiquées par une étoile, c'est le cas par exemple des tâches T9, T11, T20. Une autre caractéristique dynamique des tâches dont le nombre d'occurrence de leur exécution dépend

du nombre de points d'intérêt est indiquée par l'utilisation du facteur de ressource « *n* » que l'on note sur les tâches T9, T10 et T30.

Pour les techniques de prédiction des caractéristiques incertaines des tâches, nous avons étudié la tâche de recherche de points d'intérêt. Les temps d'exécution mesurés (sur un FPGA Virtex 5) sont donnés pour des échantillons représentatifs et présentés dans la figure 5.6 [Lefebvre, 2012]. Comme nous pouvons la remarquer, la distribution des valeurs est aléatoire en particulier dans la première échelle. Le temps écoulé par les tâches de recherche de points d'intérêt (T9, T11, T18, T20, T27 et T29) dans une image dépend du nombre de points d'intérêt que cette image contient. Le temps d'exécution est en corrélation avec le nombre de points d'intérêt trouvé, ce nombre est également aléatoire comme le montre la figure 5.7. La figure 5.7 montre le nombre de points d'intérêt présents dans l'image correspondante sur une séquence de plus de 5000 images. Elle correspond au résultat de la tâche de recherche T18 de l'échelle 3. L'évolution du nombre de point d'intérêt est similaire pour les autres tâches de recherche {T20, T9, T11, T27 et T29} des différentes échelles.

Tableau 5.3. Identification des tâches de l'application de vision robotique

Tâches		**Tâches permanentes**	**Tâches aléatoires**
Gradient	T1	T1, T2, T13, T14, T15, T16, T17, T18, T19, T20, T21	T1, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T22, T23, T24, T25, T26, T27, T28, T29, T30
Subsampling	T2		
Oversampling	T4		
Gauss1	T3, T5, T13, T14, T22, T24		
Gauss2	T15, T6, T25		
DoG	T7, T8, T16, T17, T23, T26		
Search	T9, T11, T18, T20, T27, T29		
Extract	T10, T12, T19, T21, T28, T30		

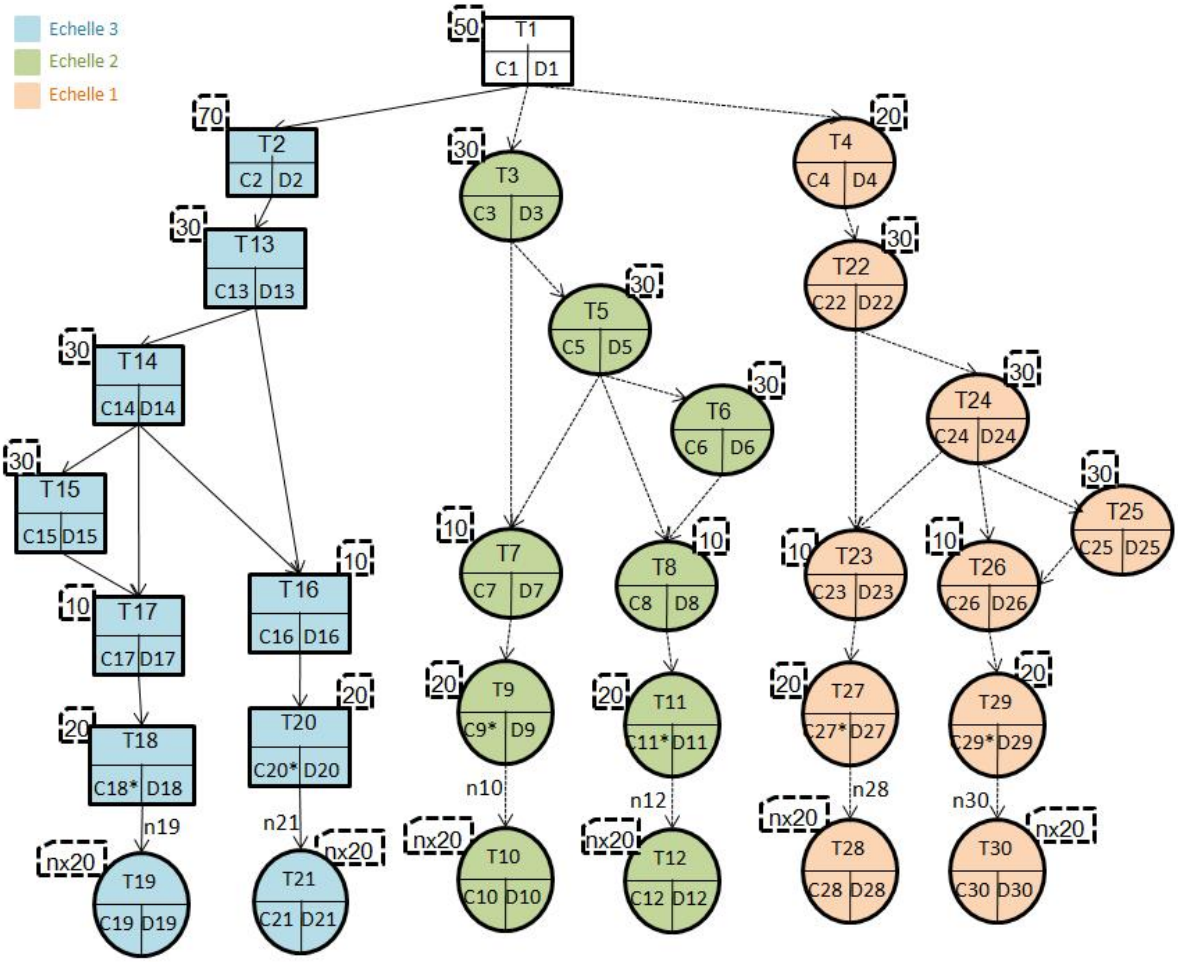

Figure 5.5. Modèle proposé pour l'application de vision robotique

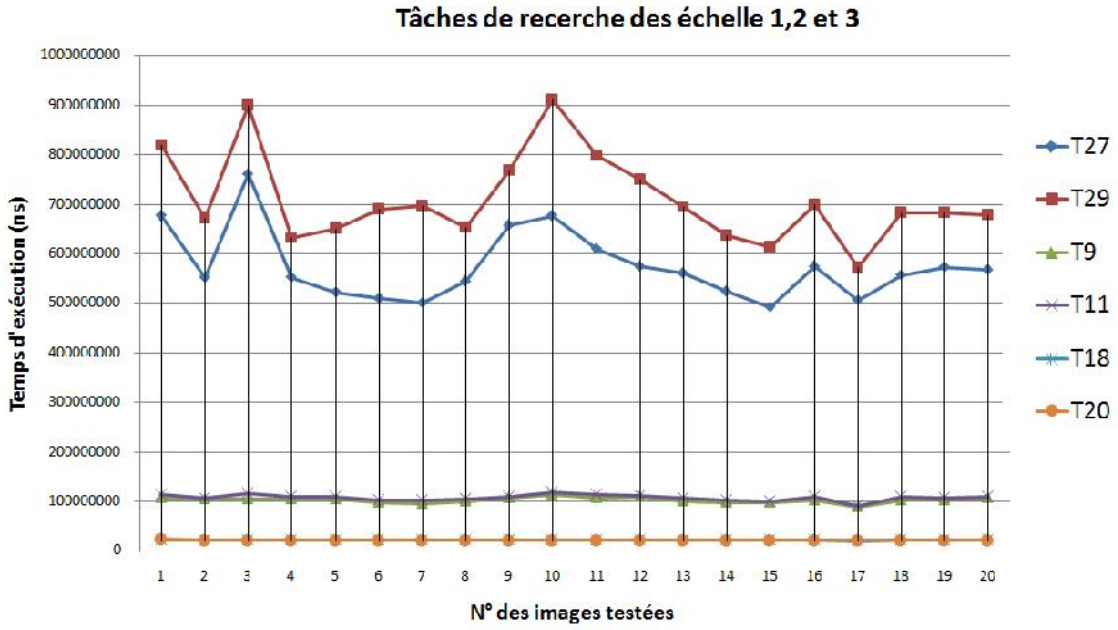

Figure 5.6. Temps d'exécution mesuré pour la tâche de recherche

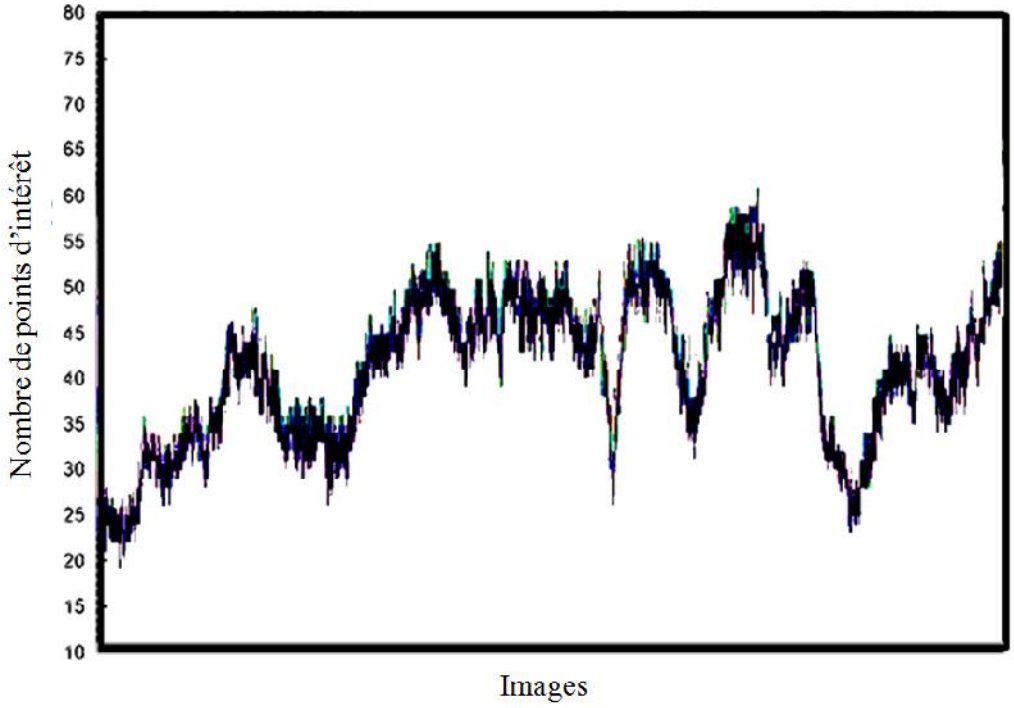

Figure 5.7. Nombre de points d'intérêt détectés dans une séquence d'image

3.3. Prédiction

Nous avons comparé plusieurs techniques pour prédire le temps d'exécution de la tâche recherche de points d'intérêt. La comparaison est basée sur l'estimation d'erreur définis comme suit:

$$E = \sum_i \frac{\text{Temps d'exécution réel (i)} - \text{Temps d'exécution estimé (i)}}{\text{Temps d'exécution réel (i)}} \; x\,100 \qquad (3)$$

où i est un entier qui varie de 0 à n, et représente le numéro d'une instance d'exécution de la tâche concernée.

Nous allons nous concentrer sur la fonction de prédiction correspondant à la tâche de recherche de l'échelle 1 car elle présente des distributions très aléatoires (figure 5.6). Nous avons testé quatre méthodes fondées sur des mesures statistiques obtenues en ligne :

1. Première méthode (a) qui utilise la valeur moyenne des trois dernières durées d'exécution réelles.

2. Deuxième méthode (b) qui utilise la moyenne des trois dernières durées multipliées par des poids différents. Les poids de pondération sont déterminés expérimentalement en se basant sur les données testés afin de minimiser l'erreur d'estimation. Dans cette technique de régression non paramétrique, les poids supérieurs sont attribués aux valeurs historiquement proches tandis que les poids les plus faibles à ceux plus éloignées.

3. Troisième méthode (c) qui prend le maximum des trois dernières valeurs réelles.

4. La quatrième méthode (d) qui prend la dernière valeur majorée de 5%.

Nous remarquons que toutes les méthodes conduisent, sur les vingt images testées, à une surestimation du temps d'exécution. Comme nous pouvons le voir sur la figure 5.8, pour la première méthode, après quelques périodes les valeurs prédites deviennent à peu près constantes. Dans le cas de seconde méthode (figure 5.9), il y a une correspondance retardée entre le graphe d'exécution réelle et celle prédite. La troisième méthode de prédiction (figure 5.10) est plus pessimiste et produit plus de surestimations. Et enfin la quatrième méthode (figure 5.11) est proche de la seconde avec une prévision plus pessimiste. Comme le montre la figure 5.12, le plus grand taux d'erreur est trouvé pour la première (a) et la troisième (c) méthodes. Pour la méthode (d), et même avec une augmentation de 1% de la dernière valeur mesurée, l'erreur est toujours supérieure à celle de la seconde méthode (b). Par conséquent, le plus faible pourcentage d'erreur obtenu est celui lié à la méthode de la figure 5.9 utilisant la moyenne pondérée des trois dernières valeurs.

L'erreur moyenne entre les résultats estimés et mesurés est d'environ 0,4% pour les échelles de fréquence basse et moyenne et environ 2,45% pour l'échelle de haute fréquence. Ces résultats montrent que la technique basée sur une moyenne pondérée des valeurs de temps d'exécution fonctionne avec une précision de près de 98%. Même pour l'estimation du nombre de points d'intérêt, voir la figure 5.7, l'erreur d'estimation moyenne de la même technique est de 0,406% pour les 5000 images. Toutefois, une erreur moyenne nulle ne garantit pas une bonne gestion des ressources. Comme nous pouvons le voir sur la forme des graphiques de la figure 5.8 jusqu'à la figure 5.11, certaines méthodes présentent plusieurs surestimations conduisant à la surexploitation des ressources, tandis que d'autres ont des sous-estimations qui conduisent à un surcoût de temps pour les corriger. Pour l'estimation du nombre de points d'intérêt, voir la figure 5.7, nous avons calculé l'erreur d'estimation sur plus

de 1000 images successives pour les méthodes (b) et (c). La première technique donne moins de surestimation que la seconde et diminue le nombre des ressources allouées inutilement. La seconde méthode présente l'avantage d'avoir une surestimation relativement acceptable de 6,5% avec une faible sous-estimation de 21% offrant ainsi un meilleur compromis entre la bonne gestion des ressources et le surcoût des estimations erronées. Pour ce dernier cas et pour l'exécution en mode lent, l'estimation des ressources nécessaires pour les tâches à nombre de ressources dynamiques permet de diminuer de 89% le surcoût de reconfiguration (voir tableau 5.4).

Tableau 5.4. Coût de reconfiguration en mode lent

	Coût reconfigurations (#Tclk)
OLLAF sans prédiction	56852400
OLLAF avec prédiction	5754496
Gain (%)	89%

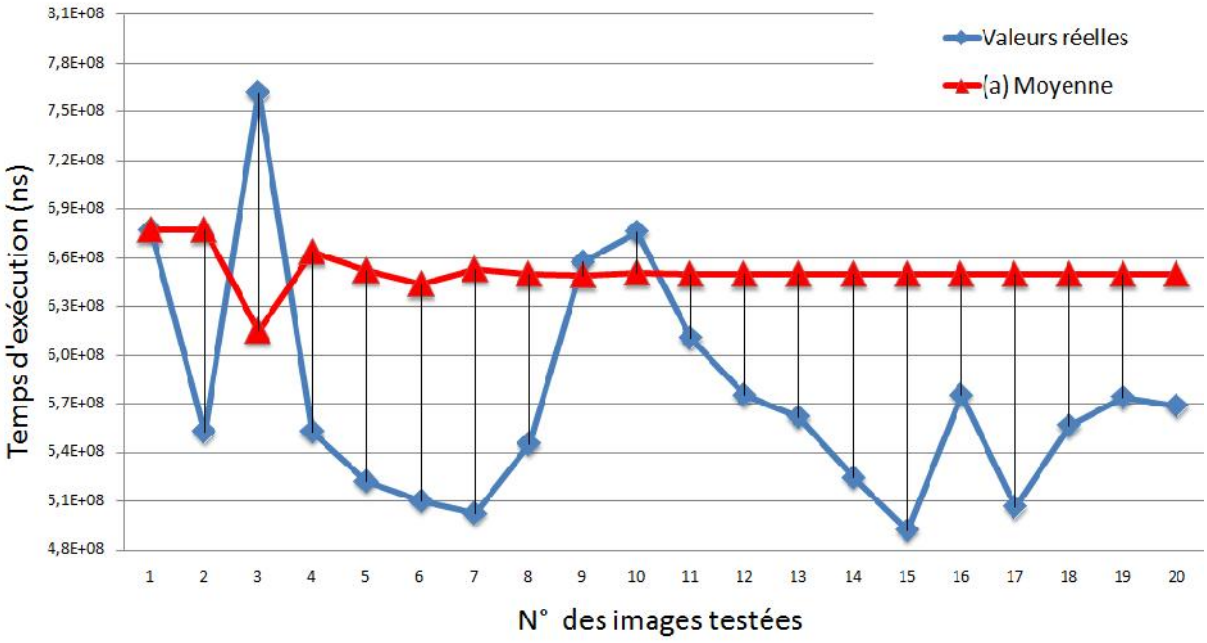

Figure 5.8. Comparaison entre valeurs réelles du durée d'exécution de la tâche T27 et celles estimées par la méthode (a)

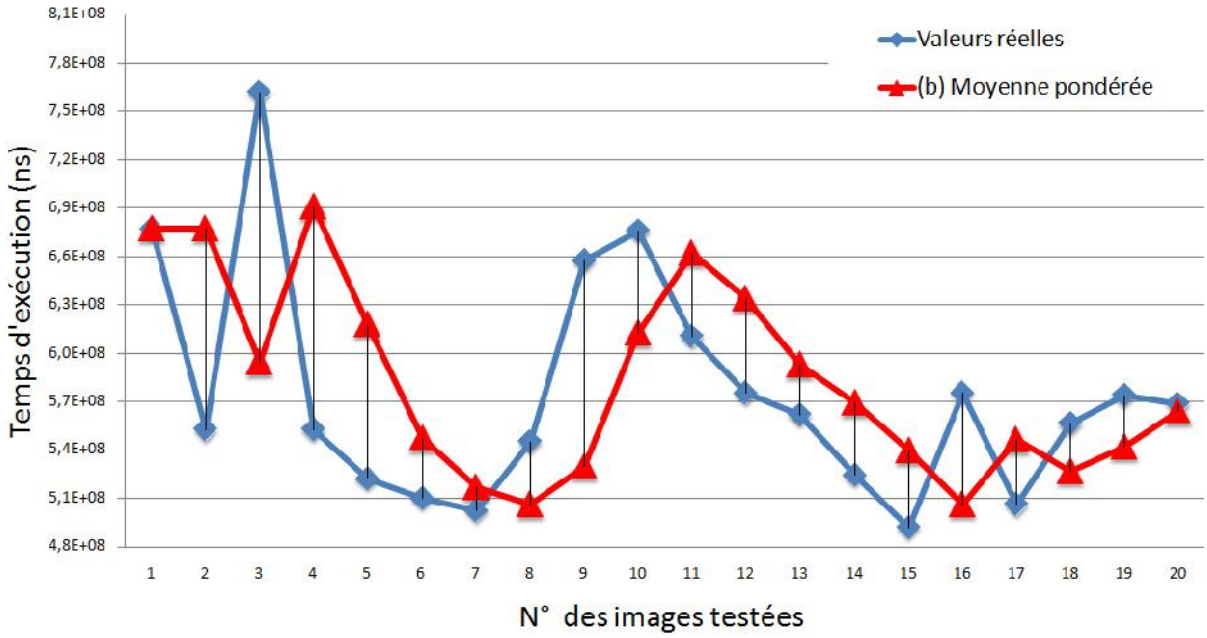

Figure 5.9. Comparaison entre valeurs réelles du durée d'exécution de la tâche T27 et celles estimées par la méthode (b)

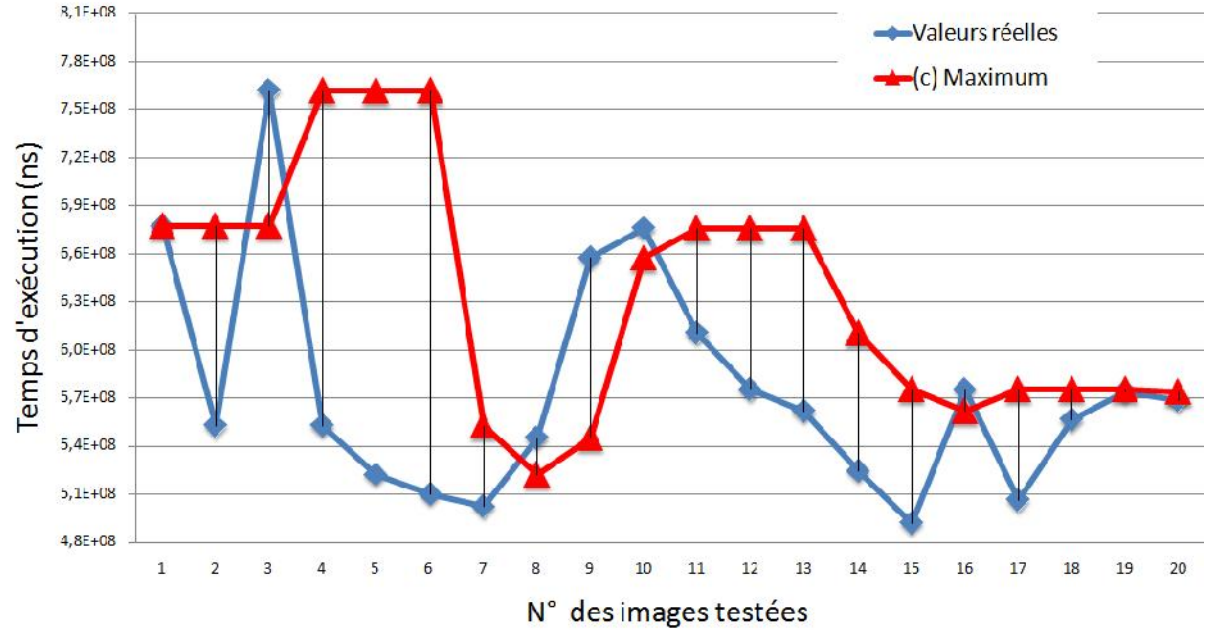

Figure 5.10. Comparaison entre valeurs réelles du durée d'exécution de la tâche T27 et celles estimées par la méthode (c)

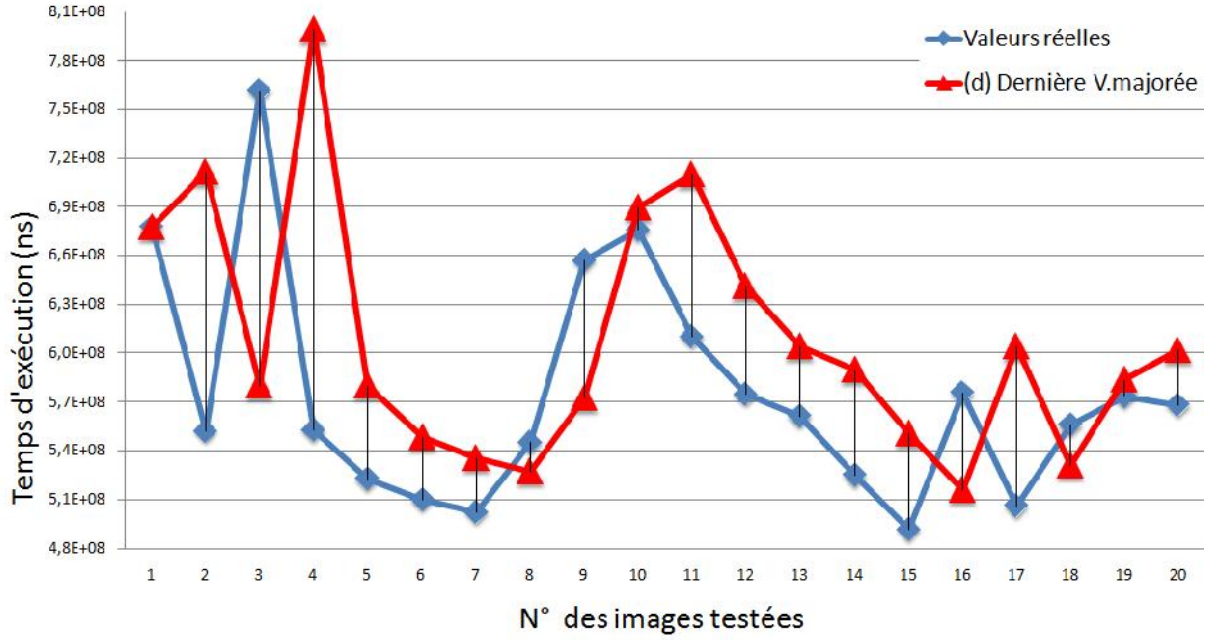

Figure 5.11. Comparaison entre valeurs réelles du durée d'exécution de la tâche T27 et celles estimées par la méthode (d)

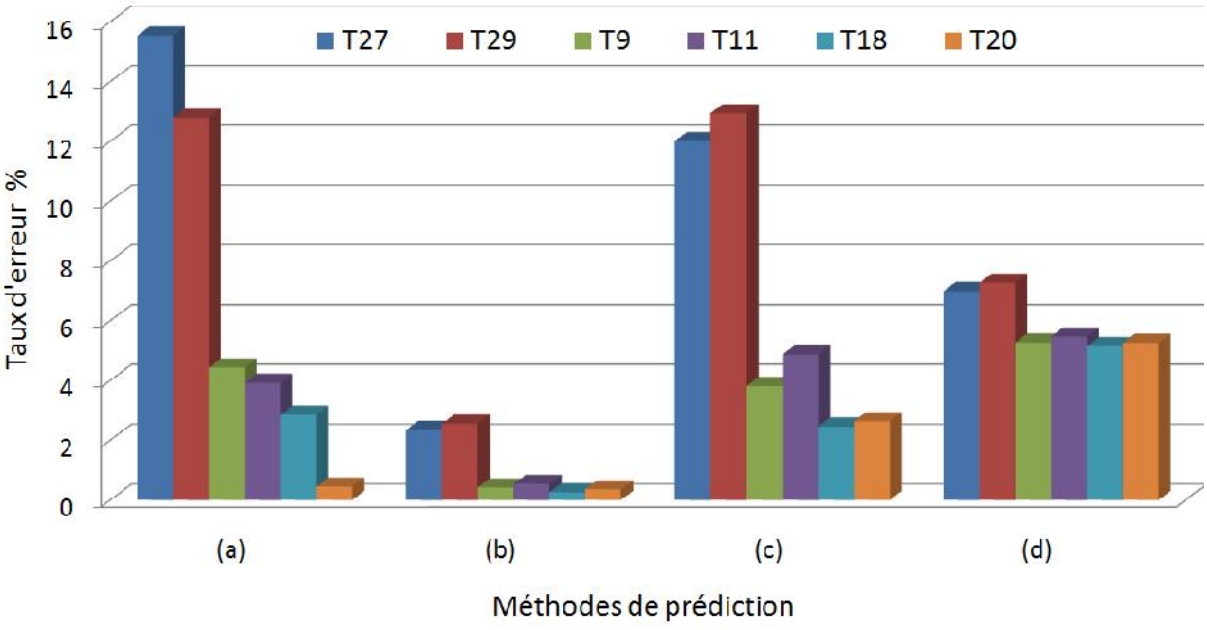

Figure 5.12. Taux d'erreur de prédiction des différentes méthodes d'estimations du temps d'exécution présenté dans la figure 5.6

4. Application de synthèse 3D

4.1. Présentation et modélisation

Nous avons également appliqué notre approche sur une application de synthèse d'image 3D [Loukil et al., 2011]. La chaîne de production d'une image 3D est appelée pipeline graphique. Elle est formée par l'ensemble des opérations nécessaires pour afficher un objet 3D regardé depuis une position et avec une orientation donnée. La figure 5.13 montre le graphe de l'application selon le modèle GMVDS. L'entrée de ce graphe est un ensemble des coordonnées locales des sommets des divers polygones qui constituent l'objet 3D. Lors du calcul du rendu 3D d'une scène, plusieurs objets peuvent être décodés indépendamment à l'intérieur d'une trame. Ce nombre d'objets ainsi que leurs caractéristiques peuvent varier d'une trame à une autre. Chaque nouvelle trame peut contenir un scénario très différent par rapport au précédent avec un nombre d'occurrence de tâches variable (tel que T9), et des tâches de durée d'exécution différents (par exemple le cas de T1).

Nous considérons une scène où deux types d'objets, (cube et cylindre) apparaissent puis disparaissent au cours de douze trames successives. Le tableau 5.5 donne les détails des différentes trames en ce qui concerne : les objets détectés, le nombre de polygones, les délais d'exécution variables pour les tâches T1 et T6 et le nombre d'occurrence estimé pour les tâches T6, T9 et T10. Un nombre à caractère gras indique une valeur sous-estimée. Les mauvaises valeurs estimées conduisent à un surcoût dû au temps de transfert des contextes des tâches.

4.2. Analyse de la prédiction

Pour le cas d'exécution de notre scène précédemment considérée, nous avons choisi la méthode d'estimation par la moyenne pondérée qui donne un meilleur résultat. Le coût en temps total est d'environ 132,54µs. En procédant à la prédiction des temps d'exécution des tâches dynamiques, ceci offre un taux de réduction de 14% du temps total d'exécution. Sinon, et sans prédiction, l'ordonnanceur doit mettre à jour les contextes des tâches dynamiques chaque fois qu'ils changent. Ceci fait au total 662,7µs ce qui est cinq fois supérieur que lors de l'utilisation de la prédiction.

Tableau 5.5. Caractéristiques des tâches d'une scène 3D

Objets	Nbre de polygones	C1*	C6*	N6	N9	N10
cube1	12	2	3	1	1	1
+ cyld1	52	**2**	**3**	**1**	**1**	**1**
+ cyld1 + cub2	100	**3**	**4**	3	3	3
+ cyld1 + cub2 + cub3	208	**4**	**5**	4	4	4
+ cyld1+ cub2 + cub3 + cyld2	328	5	6	5	5	5
+ cyld1+ cub2 + cub3 + cyld2 + cub4	520	**6**	**8**	6	6	6
cyld1+ cub2 + cub3 + cyld2 + cub4	508	6	8	7	7	7
- cyld2	388	5	7	6	6	6
- cub3	280	4	6	4	4	4
- cub4	88	3	4	3	3	3
- cyld1	48	**2**	**3**	2	2	2

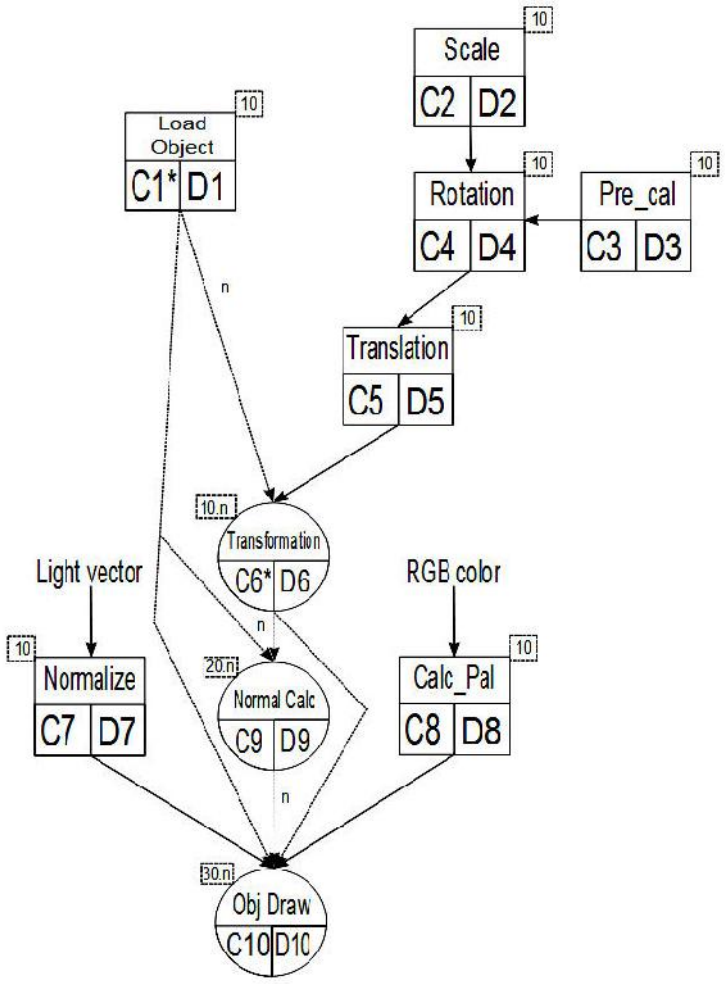

Figure 5.13. Modélisation de l'application de synthèse 3D

5. Application de traitement du flux de visioconférence

5.1. Présentation

Une autre application multimédia traitant le flux des diapositives dans une application diaporama est considérée. Présentée dans [Ouni et al., 2009], cette application porte sur le codage/décodage des diapositives et vise à effectuer un niveau de compression maximal sans dégrader la qualité des images ni le temps de traitement global. Un traitement d'image non-standard est alors proposé. La solution prend en compte deux caractéristiques différentes du flux des diapositives : les images fixes et les animations, tout en respectant un bon compromis entre le traitement rapide, la complexité opérationnelle et la qualité visuelle.

Comme le montre la figure 5.14, la technique commence par détecter les transitions entre les diapositives par des comparaisons successives entre les images reçues et une référence. Ensuite, il est procédé à l'extraction des caractéristiques des différentes diapositives : images statiques ou des portions d'animation. Puis vient un codage adaptatif par régions locales. La technique de codage dépend des caractéristiques du contenu de la région. Pour les régions de zone de texte et de bord, la compression est sans perte, tandis que pour les régions en tons continus, y compris la photo et le graphisme, la compression avec perte est tolérée. Tous les blocs compressés sont envoyés à un récepteur qui les décode et les restitue dans le flux en fonction de leur numéro d'index.

5.2. Analyse de la prédiction

La figure 5.15 montre le modèle proposé du processus d'encodage du flux de visioconférence liés aux diapositives. Nous avons considéré une présentation de conférence composée de 45 diapositives et qui dure 20 min. La résolution des trames est de 800x600 pixels et la majorité des diapositives contiennent différents effets d'animation et des transitions. On procède à l'estimation du nombre de blocs modifiés d'une trame à l'autre. Pour avoir une meilleure estimation, nous avons essayé d'exploiter la particularité de traitement de cette application. En effet, s'il y a une suite de trois trames identiques alors nous ne procédons plus à l'estimation jusqu'à la prochaine détection de nouveaux blocs à traiter. L'algorithme d'estimation utilisé est la moyenne pondérée des trois dernières mesures réelles. Par rapport aux autres algorithmes (lissage exponentiel et le maximum des trois dernières valeurs réelles),

cette méthode offre une meilleure prédiction avec moins d'erreur de surestimation et 17% de trames sous-estimées. Le surcoût de configuration des tâches T8 et T9 est amélioré avec la prédiction qui fait gagner jusqu'à 37% par rapport au cas sans prédiction du nombre de blocs modifiés.

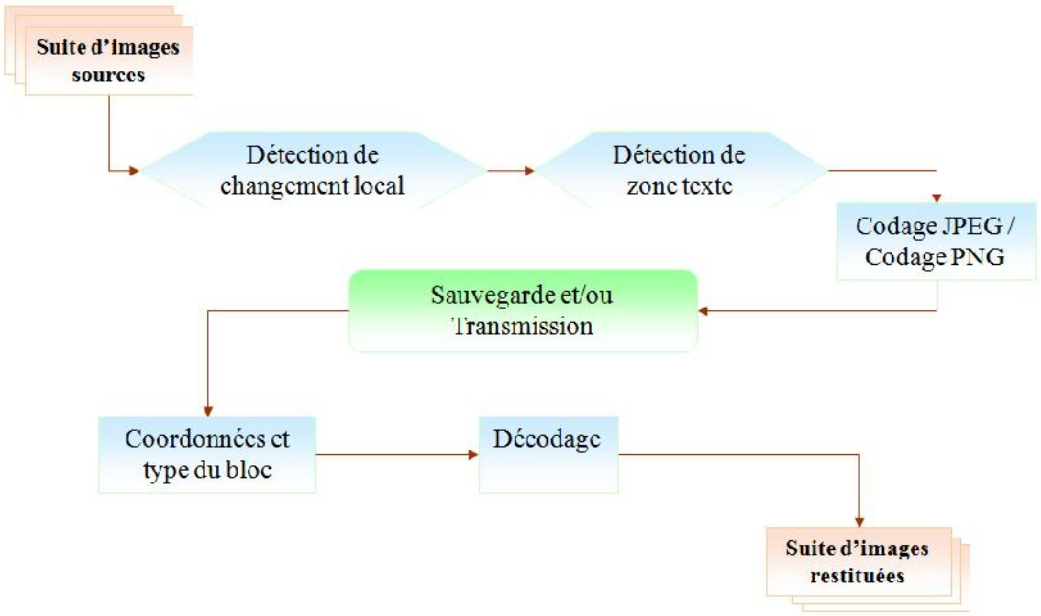

Figure 5.14. Organigramme de la méthode proposée de traitement des diapositives

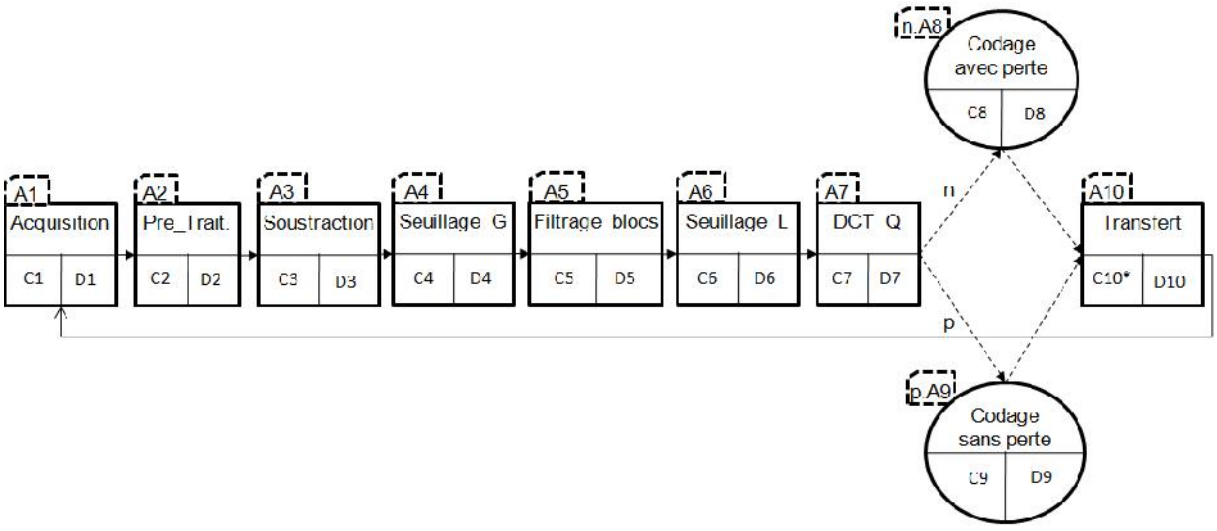

Figure 5.15. Modèle proposé pour l'application de codage des diapositives

6. Conclusion

Dans ce chapitre, nous avons détaillé les différentes expérimentations mise en œuvre pour la validation de l'approche de modélisation et d'ordonnancement prédictif. Cette validation a été faite à travers des applications multimédia dynamiques.

La première étape a consisté en la mise en place du modèle pour nos applications. Cette partie englobe les étapes de caractérisation de chaque tâche : temps d'exécution, nombre d'occurrence, nombre de ressources. Nous avons présenté quelques scénarios d'utilisation de ces applications pour la validation du fonctionnement de l'approche d'ordonnancement proposée. Ces expériences montrent que l'approche proposée a permis de réduire jusqu'à 18% le temps d'exécution. En outre, l'utilisation des heuristiques de prédiction a permis de réduire la surcharge de reconfiguration de 89%. Ce taux de réduction pourrait être plus important s'il est possible de réutiliser plus de tâches entre les différentes itérations.

Conclusion générale & perspectives

Les applications pressenties dans le futur, partagent quatre caractéristiques majeures. Elles nécessitent une capacité de calcul accrue, nécessitent la prise en compte du temps réel, représentent un pas important en terme de complexité en comparaison avec les applications d'aujourd'hui, et devront être capables de supporter la nature dynamique du monde réel.

Une architecture reconfigurable dynamiquement à grain fin (FGDRA) peut être vue comme une nouvelle évolution des FPGA d'aujourd'hui, visant à supporter des applications temps réel à la fois complexes et fortement dynamiques, tout en fournissant une puissance de calcul potentielle comparable due à la possibilité d'optimiser l'architecture dynamiquement à un niveau de granularité très fin. D'autre part, ces systèmes doivent fonctionner dans des conditions souvent difficiles, perturbantes ou aléatoires. Tous ces paramètres « dynamiques » ne sont pas pris en compte dans les méthodes de modélisation existantes. Pour rendre ce type d'architecture utilisable pour les développeurs d'applications, la complexité doit être abstraite. De ce fait, les méthodes de modélisation classiques doivent être améliorées par d'autres techniques afin de surmonter ces problèmes. Elles doivent permettre la conception de systèmes performants pour pouvoir traiter les applications complexes et flexibles qui s'adaptent à l'environnement externe variable et respecter les contraintes imposées par les ressources du système, l'environnement externe et l'utilisateur.

Cette thèse propose une approche de modélisation et d'ordonnancement pour répondre aux contraintes de durée d'exécution temps réel. Cette méthode prend en compte les caractéristiques dynamiques du système (application et architecture). Elle se compose de deux étapes : l'une se fait hors ligne lors de la conception du système et l'autre en ligne, elle intervient au cours du fonctionnement du système:

- L'étape de caractérisation hors ligne permet de modéliser l'application cible et d'extraire les différents caractéristiques dynamiques qu'elle présente.

- L'étape en ligne consiste à mettre en place un ordonnanceur prédictif qui permet de suivre en ligne l'évolution des paramètres dynamiques et de reconfigurer le système

selon les circonstances rencontrées. Cette étape est formée par une phase de prédiction et d'ajustement.

Cette méthode a été implémentée sous forme d'un exécutif qui peut être intégré à un système d'exploitation. Elle est validée à travers divers applications multimédia.

Le travail effectué dans cette thèse peut être étendu et amélioré dans plusieurs axes. Le premier thème que nous proposons, est la validation de l'approche sur des architectures reconfigurables dynamiquement autre qu'OLLAF (tel que les FPGA de type Xilinx). Bien entendu de nouveaux facteurs doivent être pris en compte tel que le coût de la reconfiguration du système en termes de temps d'exécution. Nous proposons aussi d'étudier d'autres applications multimédia qui montrent beaucoup de dynamicité. Le deuxième thème concerne l'extension de cette approche pour qu'elle supporte les architectures multiprocesseurs MPSoC, dans ce cas des questions se posent sur l'affectation de la charge de travail à chaque application, l'affectation des tâches à un processeur, la reconfiguration de l'architecture du système et bien évidemment le problème d'ordonnancement.

Bibliographie

[Abdelhalim 2008] Maaita, Adi Abdelhalim. « Techniques for Enhancing the Temporal Predictability of Real-Time Embedded Systems Employing a Time-Triggered Software Architecture ». Thèse de Doctorat, Université de Leicester, United Kingdom, Septembre 2008.

[Abel, 2006] Nicholas ABEL, "Outils et méthodes pour les architectures reconfigurables dynamiquement à grain fin. Synthèse et gestion automatique des flux de données.", Thèse de Doctorat en Traitement des images et du signal, 19 mai 2006, Université de Cergy-Pontoise, France.

[Altera, 2013] Altera. Stratix-V Device Handbook. Technical Document, volume 1, June 2013, (www.altera.com).

[Altera, 2011] Altera. Stratix-III Device Handbook. Technical Document, volume 1, March 2011, (www.altera.com).

[Altera, 2007] Altera. Stratix-II Device Handbook. Technical Document, volume 1, May 2007, (www.altera.com).

[Amano, 2006] H. Amano. "A Survey on Dynamically Reconfigurable Processors". IEICE transactions on Communications, E89-B(12) :3179 – 3187, December 2006.

[Anderson et al., 2005] Anderson, J. H.; Bud, V. & Devi, U. C. (2005), An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems, in 'Proceedings of the 17th Euromicro Conference on Real-Time Systems', IEEE Computer Society, Washington, DC, USA, pp. 199--208.

[Andersson, 2005] Andersson, J. “Modeling the Temporal Behavior of Complex Embedded Systems - A Reverse Engineering Approach”, Thèse de Doctorat, , juin 2005, Université de Malardalen, Swisse.

[ATMEL, 2013] Atmel AT40K Series FPGA Datasheet. Technical report, Atmel Corporation, January 2013.

[ATMEL, 2002] ATMEL. 5K - 50K Gates Coproessor FPGA with FreeRAM. Technical report, ATMEL Inc., 2002.

[Audsley et al., 1991] Audsley, N.; Burns, A.; Richardson, M. F. & Wellings, A. J. (1991), Hard Real-Time Scheduling: The Deadline-Monotonic Approach, in 'in Proc. IEEE Workshop on Real-Time Operating Systems and Software', pp. 133--137.

[Badouel et Oliver, 1998] E. Badouel and J. Oliver. "Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes". In Proc. of a workshop within the 19th Int’l Conf. on Applications and Theory of Petri Nets, 1998

[Balarin et al., 1998] Balarin, F.; Lavagno, L.; Murthy, P. & Sangiovanni-vincentelli, A. (1998), 'Scheduling for Embedded Real-Time Systems', IEEE Des. Test 15(1), 71--82.

[Baumgarte et al., 2003] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt. PACT XPP—A Self-Reconfigurable Data Processing Architecture. J. Supercomput., 26(2): 167–184, 2003.

[Ben Abdallah, 2009] Faten BEN ABDALLAH MANAI, "Étude et optimisation de l'interaction processeurs-architectures reconfigurables dynamiquement", Thèse de Doctorat, 20 octobre 2009, Univesité de Rennes 1, France.

[Benkermi 2007] Imen Benkermi, « Modèle et algorithme d'ordonnancement pour architectures reconfigurables dynamiquement », Ph.D. Thesis, Université de Rennes 1, ENSSAT, IRISA, Janvier 2007.

[Benoit et al., 2004] Pascal Benoit, Gilles Sassatelli, Lionel Torres, Michel Robert, Gaston Cambion, "Architectures Reconfigurables : les processeurs du futur", journées des doctorants de l'école doctorale I2S, 2 mars 2004, Montpellier, France, CD-ROM proceedings.

[Bidot, 2005] «A General Framework Integrating Techniques for Scheduling under Uncertainty », Ph.D. Thesis, Institut National Polytechnique de Toulouse, France, Novembre 2005.

[Billaut et al., 2005] Billaut, J.; Moukrim, A. & Sanlaville, E.Billaut, J.; Moukrim, A. & E., S., ed. (2005), *Flexibility and Robustness in Scheduling*, Hermes.

[Bonamy et al., 2012] Bonamy, R.; Pham, H.-M.; Pillement, S. & Chillet, D. (2012), UPaRC - Ultra-fast power-aware reconfiguration controller, *in* 'DATE', pp. 1373-1378.

[Bossuet, 2004] Lilian BOSSUET, "Exploration de l'Espace de Conception des Architectures Reconfigurables", Thèse de Doctorat, 10 septembre 2004, Université de Bretagne Sud, France.

[Brown et al., 1992] Brown, S., Francis, R., Rose, J., and Vranesic, Z., Field Programmable Gate Arrays. Boston, USA: Kluwer and Acad. Publishers, 1992.

[Buttazzo, 2004] Buttazzo, G. C. (2004), *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*, Springer-Verlag TELOS, Santa Clara, CA, USA.

[Cardoso and Hübner, 2011] Cardoso, J. & Hübner, M. (2011), *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*, Springer.

[Chang et al., 1999] Chang, H.; Cooke, L.; Hunt, M.; Martin, G.; McNelly, A. J. & Todd, L. (1999), *Surviving the SOC revolution: a guide to platform-based design*, Kluwer Academic Publishers, Norwell, MA, USA.

[Choi et al., 2012] Choi, S. H. & Wang, K. (2012), 'Flexible flow shop scheduling with stochastic processing times: A decomposition-based approach', *Comput. Ind. Eng.* **63**(2), 362--373.

[Chtourou, 2007] Sofien CHTOUROU, "Méthodologies de synthèse de réseaux de neurones pour applications de traitement de signal adaptatif et implémentation sur circuits reconfigurables dynamiquement", Thèse de Doctorat, 04/06/2007, Institut National des Sciences Appliquées de Rennes, France.

[Compton et Hauck, 2000] Z. Li, K. Compton, and S. Hauck, “Configuration caching techniques for FPGA”, in Proceedings of the IEEE Symposium on FPGA for Custom Computing Machines (FCCM’00), Napa Valley, Calif, USA, April 2000.

[Compton et Hauck, 2002] Compton, K. & Hauck, S., “Reconfigurable computing: a survey of systems and software”, *ACM Computing Survey*. **34**(2), 171—210, (2002).

[Cortes et al., 1999] L. Alejandro Cortés, P. Eles and Z. Peng, "A Survey on Hardware/Software Codesign Representation Models", SAVE Project, Dept. of Computer and Information Science, Linköping University, Linköping, June 1999.

[Cortes, 2005] L. Alejandro Cortes, "Verification and Scheduling Techniques for Real-Time Embedded Systems", Ph. D. Thesis No. 920, Dept. of Computer and Information Science, Linköping University, March 2005.

[Darouich, 2008] Darouich M., Guyetant S., Chevobbe S., « Service de configuration prédictif pour plateforme multicoeur reconfigurable hétérogène », RenPar'18 / SympA'2008 / CFSE'6, Fribourg 2008.

[Danne et Platzner, 2005] K. Danne and M. Platzner. « Periodic Real-Time Scheduling for FPGA Computers ». In Proc. of the 3rd International Workshop on Intelligent Solutions in Embedded Systems (WISES), pages 117–127. IEEE Computer Society, Mai 2005.

[Davenport et Beck, 2000] Andrew J. Davenport and J. Christopher Beck. « A survey of techniques for scheduling with uncertainty », Technical report, IBM and Ilog, 2000.

[David et al., 2002] David. R, Chillet. D, Pillement. S, Sentieys. D, “DART: A Dynamically Reconfigurable Architecture dealing with next Generation Telecommunications Constraints”, in Reconfigurable Architecture Workshop, 2002.

[David, 2003] Raphaël David. "Architecture reconfigurable dynamiquement pour applications mobiles", Thèse de Doctorat, juillet 2003, Univesité de Rennes 1, France.

[David et al, 2005] Raphaël David, Dominique Lavenier, and Sébastien Pillement. « Du micro-processeur au circuit FPGA : une analyse sous l’angle de la reconfiguration ». *Technique et Science Informatiques*, 24(4) :395–422, 2005.

[Duhem et al., 2011] Duhem, F.; Muller, F. & Lorenzini, P. (2011), FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA, *in* 'ARC', pp. 253-260.

[Duranton et al., 2009] Marc Duranton, Sami Yehia, Bjorn De Sutter, Koen De Bosschere, Albert Cohen, Babak Falsa, Georgi Gaydadjiev, Manolis Katevenis, Jonas Maebe, Harm Munk, Nacho Navarro, Alex Ramirez, Olivier Temam et Mateo Valero. "The HIPEAC vision". Rapport technique, Network of Excellence on High Performance and Embedded Architecture and Compilation (HIPEAC), 2009.

[Edgar et Burns, 2001] Edgar, S. & Burns, A. (2001), Statistical Analysis of WCET for Scheduling., *in* '22nd IEEE Real-Time Systems Symposium (RTSS)', IEEE Computer Society, London, UK, pp. 215-224.

[Edwards et al., 1997] Stephen Edwards, Luciano Lavagno, Edward A. Lee, and Alberto Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis", PROCEEDINGS OF THE IEEE, VOL. 85, NO. 3, MARCH 1997, PP. 366–390.

[Engblom et al., 2003] Engblom, J.; Ermedahl, A.; Sjµdin, M.; Gustafsson, J. & Hansson, H. (2003), 'Worst-case execution-time analysis for embedded real-time systems', International Journal on Software Tools for Technology Transfe (STTT) 4, 437-455.

[Erschler et al., 2001] Erschler, J. & Grabot, B., Hermès - Lavoisier, ed., (2001), *Gestion de production: fonctions, techniques et outils*, Hermès Science Publications, chapter 5 Prévision et planification, pp. 113-155.

[Esswein, 2003] Carl Esswein. « Un apport de flexibilité séquentielle pour l'ordonnancement robuste ». Thèse de doctorat, Université François Rabelais Tours, 2003.

[Fishwick, 2007] P.A Fishwick, "Handbook of dynamic system modeling", Chapman & Hall/CRC Computer and Information Science Series 2007.

[Fortier et al., 2003] Fortier, P. J. & Michel, H. (2003), Computer Systems Performance Evaluation and Prediction, Butterworth-Heinemann, Newton, MA, USA.

[Garcia et al., 2007] S. Garcia, J. Prevotet, and B. Granado, "Hardware task context management for fine grained dynamically reconfigurable architecture," in Proceedings of the Workshop on Design and Architectures for Signal and Image Processing (DASIP '07), Grenoble, France, November 2007. 38, 91, 92

[Garcia et al., 2008] Garcia, S. & Granado, B. (2008), OLLAF: a Dual Plane Reconfigurable Architecture for OS Support, *in* 'International Design and Test Workshop (IDT)', pp. 282 - 287.

[Garcia et al., 2009] Garcia, S. & Granado, B. (October 2009), "OLLAF: a Fine Grained Dynamically Reconfigurable Architecture for OS Support", EURASIP Journal on Embedded Systems - Special issue on design and architectures for signal and image processing 2009, 10:2--10:2.

[Garcia, 2012] Samuel Garcia, « Architecture reconfigurable dynamiquement à grain fin pour le support d'un système d'exploitation temps réel », Thèse de Doctorat, Université de Pierre et Marie Curie - Paris 6, Spécialité Informatique, 14 Mai 2012.

[Ghaffari, 2006] Fakhreddine Ghaffari, « Partitionnement en ligne d'applications flots de données pour des architectures temps réel auto-adaptatives », Thèse de Doctorat, Université de Nice-Sophia Antipolis, Spécialité Electronique, 30 Novembre 2006.

[Giard, 2003] Giard, V. (2003), *Gestion de la production et des flux*, Economica, Paris.

[Gubner, 2006] Gubner, J. A. (2006), *Probability and Random Processes for Electrical and Computer Engineers*, Cambridge University Press, New York, NY, USA.

[Harrath 2003] Youssef Harrath, « Contribution à l'ordonnancement conjoint de la production et de la maintenance : Application au cas d'un Job-Shop ». Thèse de Doctorat, Université de Franche-Comté, Spécialité Automatique et Informatique, 16 décembre 2003.

[Hartenstein, 2001] R. Hartenstein, "A decade of reconfigurable computing : a visionary retrospective". In Proceedings of the conference on Design, automation and test in Europe (DATE'01), p. 642–649, 2001. 4, 16.

[Iverson et al., 1996] Iverson, M. A.; Ozguner, F. & Follen, G. J. (1996), "Run-Time Statistical Estimation of Task Execution Times for Heterogeneous Distributed Computing", Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing.

[Jantsch, 2003] A. Jantsch. "Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation". Morgan Kaufmann, San Francisco, CA, 2003.
[Kerzner 2003] Kerzner, H. (2003), "*Project Management: A Systems Approach to Planning, Scheduling, and Controlling"*, John Wiley & Sons.

[Kessal et al., 2006] L. Kessal, N. Abel et D. Demigny, "Traitement temps réel des images en exploitant la reconfiguration dynamique : architecture et programmation", TS traitement du signal 2006, vol. 23, no1, pp. 41-58.

[Keutzer et al., 2000] K. Keutzer et al., "System-Level Design: Orthogonalization of Concerns and Platform-Based Design". In Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions Dec 2000.

[Kienhuis, 1997] B. Kienhuis, "An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures". In Application-Specific Systems, Architectures and Processors, july 1997.

[Krishnaswamy et al., 2004] Krishnaswamy, S.; Loke, S. W. & Loke, S. W. (2004), 'Estimating Computation Times of Data-Intensive Applications', IEEE Distributed Systems Online 5.

[Ktata et al., 2010] Ktata, I.; Ghaffari, F.; Granado, B. & Abid, M. (2010), "Novel Approach for Modeling Very Dynamic and Flexible Real Time Applications", Proceedings of the 5th International Workshop on Reconfigurable Communication-centric Systems-on-Chip.

[Ktata et al., 2011] Ktata, I.; Ghaffari, F.; Granado, B. & Abid, M. (2011), 'Dynamic application model for scheduling with uncertainty on reconfigurable architectures', Int. J. Reconfig. Comput. 2011, 3:1--3:15.

[Lallet, 2008] Julien LALLET, "Mozaïc : plate-forme générique de modélisation et de conception d'architectures reconfigurables dynamiquement", Thèse de Doctorat, 26 novembre 2008, Univesité de Rennes 1, France.

[Lavagno et al., 1999] L. Lavagno, A. Sangiovanni-Vincentelli, and E. Sentovich, "Models of computation for embedded system design". In A. A. Jerraya and J. Mermet, editors, System-Level Synthesis, pages 45–102, Dordrecht, 1999. Kluwer.

[Lefebvre, 2012] Thomas Lefebvre, « Exploration architecturale pour la conception d'un système sur puce de vision robotique, adéquation algorithme-architecture d'un système embarqué temps-réel », Thèse de Doctorat, Université de Cergy-Pontoise, France, 2 juillet 2012.

[Leung, 2004] Joseph Y-T. Leung, « Handbook of Scheduling: Algorithms, Models, and Performance Analysis », Chapman & Hall/CRC Computer & Information Science Series, 2004.

[Liu et al., 1973] C.L. Liu and J.W. Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment". Journal of ACM 20(1):46-61, 1973.

[Loukil et al., 2011] Loukil, K.; Ben Amor, N. & Abid, M. (2011), 'HW/SW Partitioning Approach on Reconfigurable Multimedia System on Chip', International Journal of Engineering -ISSN 1985-2312 5; Issue: 1, 568-578.

[Merhoum et al., 2007] K. Merhoum, M. Djeghaba. « Algorithme génétique pour le problème d'ordonnancement de type job-shop ». Conférence Internationale sur la Productique, CIP'2007.

[Michael, 2008] Pinedo Michael L., « Scheduling Theory, Algorithms, and Systems », 2008 Springer Science+Business Media , ISBN: 978-0-387-78934-7.

[Moore, 1965] Moore, G. E. (1965), 'Cramming more components onto integrated circuits', *Electronics Magazine* **38**(8).

[Mtibaa et al., 2007] A. Mtibaa, B. Ouni and M. Abid, "An efficient list scheduling algorithm for time placement problem", Computers and Electrical Engineering 33 (2007) 285–298.

[Murata, 1989] Tadao Murata, "Petri nets: Properties, Analysis and Applications", Proceedings of the IEEE, 77(4):541-574, April 1989.

[Noguera et al., 2004] J. Noguera, R.M. Badia, "Multitasking on reconfigurable architectures: Microarchitecture support and dynamic scheduling", ACM Transactions on Embedded Computing Systems, Volume 3, Issue 2 (May 2004) pp. 385-406.

[Nollet et al., 2003] Nollet, V.; Coene, P.; Verkest, D.; Vernalde, S. & Lauwereins, R. (2003), Designing an Operating System for a Heterogeneous Reconfigurable SoC, *in* 'Proceedings of the 17th International Symposium on Parallel and Distributed Processing', IEEE Computer Society, Washington, DC, USA, pp. 174.1--.

[Ouni et al., 2009] OUNI, T., KTATA, I., AND ABID, M. 2009. "Adapted image-based method of slide stream processing in slideshow applications". In Proceedings of the conference on Design and Architectures for Signal and Image Processing.

[Ourari, 2011] Samia OURARI, "De l'ordonnancement déterministe à l'ordonnancement distribué sous incertitudes", Thèse de Doctorat, Université de Toulouse III, spécialité: systèmes industriels, 28 janvier 2011.

[Pailler 2006] Stéphane PAILLER, « Analyse Hors Ligne d'Ordonnançabilité d' Applications Temps Réel comportant des Tâches Conditionnelles et Sporadiques », Thèse de Doctorat, Université de Poitiers, Spécialité informatique, 19 Octobre 2006.

[Parizi et al., 2002] Parizi. H, Niktash. A, Bagherezadeh. N and Kurdahi. F, 'MorphoSys: A coarse grain reconfigurable architecture for multimedia applications', Euro-par 2002, LNCS 2004, pp.844-848.

[Peterson, 1981] J. Peterson, "Petri Net Theory and the Modeling of Systems". Englewood Cliffs, NJ: Prentice-Hall, 1981.

[Piétrac, 1999] Laurent Piétrac, «Apport de la méta-modélisation formelle pour la conception des Systèmes Automatisés de Production », Thèse de doctorat de l'école normale supérieure de Cachan, spécialité Automatique, 12 janvier 1999.

[Pillement, 2010] Sébastien Pillement, "Conception d'architectures reconfigurables dynamiquement : Du silicium au système", Habilitation à Diriger des Recherches (HDR), Université de RENNE 1, le 22 octobre 2010.

[Pinot, 2008] Guillaume Pinot, « Coopération homme-machine pour l'ordonnancement sous incertitudes », Thèse de doctorat de l'Université de Nantes, spécialité Génie Informatique, Automatique et Traitement du Signal, 14 novembre 2008.

[Rammig et Rust, 2003] Franz-Josef Rammig, Carsten Rust, "Modeling of Dynamically Modifiable Embedded Real-Time Systems", WORDS Fall 2003: 28-34.

[Resano et al., 2004] Resano J., Mozos D., Verkest D. Vernalde S. et Catthoor F. 2004. « A hybrid design-time/run-time scheduling flow to minimize the reconfiguration overhead of FPGAs ». J. Microproc. Microarchi. 28, 5--6, 291--301. 2004.

[Resano et al., 2005a] Javier Resano , Daniel Mozos , Francky Catthoor, « A Hybrid Prefetch Scheduling Heuristic to Minimize at Run-Time the Reconfiguration Overhead of Dynamically Reconfigurable Hardware ». Proceedings of the conference on Design, Automation and Test in Europe, Munich, Germany, p.106-111, March 07-11, 2005.

[Resano et al., 2005b] Resano, J. ; Mozos, D. ; Catthoor, F. et Verkest, D. « A Reconfiguration Manager for Dynamically Reconfigurable Hardware ». IEEE Design Test of Computers, v.22 n.5, p.452-460, September 2005.

[Resano et al., 2008] Resano J., Clemente J., Gonzalez C., Mozos D. et Catthoor F. « Efficiently Scheduling Runtime Reconfigurations ». Transactions on Design Automation of Electronic Systems (TODAES) 13(4): 1-12, September 2008.

[RTAI] Real Time Application Interface Official Website: www.rtai.org, Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano, Italy.

[Rust et al., 2004] Carsten Rust, Franz J. Rammig, "A Petri Net Based Approach for the Design of Dynamically Modifiable Embedded Systems". DIPES 2004: 257-266.

[Sanlaville, 2005] Sanlaville, E., "Ordonnancement sous conditions changeantes – Comment prendre en compte les variations, aléas, incertitudes sur les données", Habilitation à Diriger des Recherches (HDR), Université Blaise Pascal de Clermont-Ferrand, March 7, 2005.

[Sassatelli et al., 2002] G. Sassatelli, L. Torres, P. Beboit, T. Gill, G. Cambon and J. Galy. "Highly scalable dynamically reconfigurable Systolic Ring". In IEEE Design Automation and Test in Europe, pages 553-557, Paris (France), march 2002.

[Savourey, 2006] David Savourey, "Ordonnancement sur machines parallèles: minimiser la somme des coûts", Thèse de Doctorat de l'Université de Technologie de Compiègne, spécialité: Technologie de l'information et des systèmes, 5 décembre 2006.

[Sha et al., 1994] Sha, L.; Rajkumar, R. & Sathaye, S. (1994), 'Generalized rate-monotonic scheduling theory: a framework for developing real-time systems', Proceedings of the IEEE 82(1), 68 -82.

[Shibata et al., 2000] Y. Shibata, M. Uno, H. Amano, K. Furuta, T. Fujii, and M. Motomura, "A virtual hardware system on a dynamically reconfigurable logic device", in Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'00), Napa Valley, Calif, USA, April 2000.

[Sinnen, 2007] Oliver Sinnen. "Task Scheduling for Parallel Systems", (Wiley Series on Parallel and Distributed Computing). Wiley-Interscience, 2007.

[So et al., 2006] So, H. K.-H.; Tkachenko, A. & Brodersen, R. (2006), A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH, *in* 'Proceedings of the 4th international conference on Hardware/software codesign and system synthesis', ACM, New York, NY, USA, pp. 259--264.

[Sotskov et al., 2010] Yu.N. Sotskov, N.Yu. Sotskova, T.-C. L. & Werner, F. (2010), *Scheduling under uncertainty: Theory and Algorithms*, Belarusian Science, Minsk.

[Stankovic et al., 1998] Stankovic, J. A.; Ramamritham, K. & Spuri, M. (1998), *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*, Kluwer Academic Publishers, Norwell, MA, USA.

[Steiger et al., 2004] C.Steiger, H.Walder, M.Platzner, "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks", Computers, IEEE Transactions on Volume 53, Issue 11, Nov. 2004 Page(s): 1393 - 1407.

[Tavana, 2008] M. Tavana, "Dynamic process modelling using Petri nets with applications to nuclear power plant emergency management", Int. J. Simulation and Process Modelling (2008), Vol. 4, No. 2, pp.130–138.

[Torres et al., 2011] Torres, L. & Zhao, W. (2011), Magnetic memory (MRAM), a new area for 2D and 3D SoC/SiP design, *in* 'Proceedings of the 21st edition of the great lakes symposium on Great lakes symposium on VLSI', ACM, New York, NY, USA, pp. 429--430.

[Trung, 2005] LA Hoang Trung, « Utilisation d'ordres partiels pour la caractérisation de solutions robustes en ordonnancement », Thèse de doctorat de l'Institut National des Sciences Appliquées de Toulouse, spécialité systèmes industriels, 24 Janvier 2005.

[Vassiliadis et al., 2007] Vassiliadis, S. & Soudris, D. (2007), "Fine- and Coarse-Grain Reconfigurable Computing", Springer Publishing Company, Incorporated.

[Verdier et al., 2008] Verdier, F.; Miramond, B.; Maillard, M.; Huck, E. & Lefebvre, T. (2008), 'Using High-Level RTOS Models for HW/SW Embedded Architecture Exploration: Case Study on Mobile Robotic Vision', EURASIP Journal on Embedded Systems.

[Veysseyre, 2006] Veysseyre, R. (2006), *Aide-mémoire statistique et probabilités pour l'ingénieur*, Dunod, Paris.

[Wigley et Kearney, 2001] Wigley, G. & Kearney, D. (2001), 'The first real operating system for reconfigurable computers', *Aust. Comput. Sci. Commun.* **23**(4), 130--137.

[Wigley et Kearney, 2002] Wigley, G. & Kearney, D. (2002), 'The management of applications for reconfigurable computing using an operating system', *Aust. Comput. Sci. Commun.* **24**(3), 73--81.

[Wigley et al., 2006] Wigley, G. B.; Kearney, D. A. & Jasiunas, M. (2006), ReConfigME: a detailed implementation of an operating system for reconfigurable computing., *in* 'IPDPS', IEEE.

[Wilhelm et al., 2008] Wilhelm, R.; Engblom, J.; Ermedahl, A.; Holsti, N.; Thesing, S.; Whalley, D.; Bernat, G.; Ferdinand, C.; Heckmann, R.; Mitra, T.; Mueller, F.; Puaut, I.; Puschner, P.; Staschulat, J. & Stenstrцm, P. (2008), 'The worst-case execution-time problem—overview of methods and survey of tools', ACM Transactions on Embedded Computing Systems (TECS) 7, 36:1--36:53.

[Wilhelm et al., 2006] Wilhelm, R.Zurawski, R., ed. (2006), "Determining Bounds on Execution Times" In Handbook on Embedded Systems 2005, CRC Press.

[Xilinx, 2001] Xilinx. Virtex 2.5 V Field Programmable Gate Arrays : Product Specification. Technical report, Xilinx, 2001.

[Xilinx, 2011] XILINX inc., "Partial Reconfiguration User Guide". Rapport technique: UG702 (v13.1) March 1, 2011, (www.xilinx.com).

[Xilinx, 2013] XILINX inc., "7 Series FPGAs Configuration User Guide". Rapport technique: UG470 (v1.6) January 2, 2013, (www.xilinx.com).

[Zhuo et al., 2006] Zhuo, Y.; Li, H. & Mohanty, S. P. (2006), A Congestion Driven Placement Algorithm for FPGA Synthesis, *in* 'Proceedings of the 2006 International Conference on Field Programmable Logic and Applications (FPL), Madrid, Spain, August 28-30, 2006', pp. 1-4.

Réalisation d'un système d'exploitation pour l'architecture reconfigurable dynamiquement OLLAF

Ismail KTATA

الخلاصة: إن النظم المضمّنة لديها متطلبات هامة مثل الحد من التعقيد وتوفير جهود التنمية والتطوير. على هذه النظم أيضا أن تأخذ في الاعتبار قيود التطبيقات المتعلقة بالتوقيت، والموارد، وعلاقات الأسبقية بين المهام وغيرها من خصائص النظم العامة التي قد تتغير أثناء التنفيذ. ولتلبية هذه القيود، يتوجب على هذه النظم أن تكون قادرة على دعم الطبيعة المتغيرة للعالم الحقيقي وذلك في مرحلة مبكرة من تصميمها. في هذا الإطار تعتبر المعماريات القابلة للبرمجة وإعادة التشكيل (DRA) الحل المثالي لتلبية السلوك الديناميكي للغاية والغير قطعي للتطبيقات الحالية لأنه يوفر كلا من الأداء العالي والمرونة وقت التشغيل. وقد قدمنا في هذه الأطروحة منهجية جديدة تجمع بين النمذجة على مستوى عالي والجدولة على نوع جديد من المعماريات القابلة لإعادة التشكيل الحيوي. وبناءا على النموذج المقترح لتبيان المهام المكونة للتطبيقات، يتم تنفيذ جدولة بنهج تقديري وتنبؤي. وتهدف الطريقة المقترحة لتحسين إدارة عملية إعادة تشكيل البنية وتقليل زمن الوصول لها. وقد أثبتت النتائج التجريبية على البنية الجديدة والمدعاة OLLAF بيان المنافع وكفاءة تقنية الجدولة التي قدمناها.

Résumé : Actuellement on assiste à une émergence des applications des systèmes embarqués destinées à un large public d'utilisateurs. Ces applications sont de plus en plus complexes et diversifiées. Elles nécessitent une capacité de calcul accrue et doivent satisfaire, dans leurs exécutions, la prise en compte du temps réel. De plus, ces systèmes sur puce fonctionnent dans des conditions souvent difficiles et perturbantes. Ainsi, certaines contraintes temporelles, contraintes de ressources, contraintes de précédence ainsi que d'autres caractéristiques des systèmes généraux peuvent changer au cours d'exécution. Pour respecter leurs contraintes, ces systèmes doivent être capables de supporter la nature dynamique du monde réel depuis la modélisation de l'application jusqu'à son implémentation sur la plateforme d'exécution. Dans cette thèse une nouvelle approche combinant la modélisation haut niveau et l'ordonnancement sur une architecture reconfigurable dynamiquement de nouveau type, a été proposée. Cette approche est originale depuis ça conception en ciblant des applications fortement dynamiques et flexibles. De plus, l'ordonnanceur ainsi développé intègre un nouveau service qui est responsable de la prédiction des variables dynamiques afin d'aboutir à une meilleure exploitation de l'architecture et meilleure performance d'exécution. Des expérimentations ont été présentées sur des applications temps réel.

Abstract: Embedded systems have important requirements such as reducing complexity and saving development effort. They have also to take account of applications constraints related to timing, resources, tasks precedence relations and other characteristics of general systems that may change during execution. To meet their constraints, these systems must be capable of supporting the dynamic nature of the real world at an early phase of their design. Dynamically reconfigurable architecture (DRA) is presented as the ideal solution to satisfy the highly dynamic and non-deterministic behavior of current applications since it provides both high performance and run-time flexibility. In this thesis a new approach combining the high level modeling and scheduling on a dynamically reconfigurable architecture of a new type, has been proposed. Based on an original task graph model, the scheduling is performed by a predictive approach. The proposed method aims to better manage the reconfiguration process and minimize its latency. Experimental results based on the original DRA named OLLAF demonstrate the benefits and efficiency of our scheduling technique.

المفاتيح: الجدولة، المعماريات القابلة للبرمجة وإعادة التشكيل، بنية أولاف، النمذجة، خوارزمية الكشف عن مجريات الأمور، الديناميكية

Mots clés: Ordonnancement, architecture reconfigurable, OLLAF, modélisation, heuristiques, dynamicité.

Key-words: Scheduling, reconfigurable architectures, OLLAF, modeling, heuristics, dynamicity.