ENSSAT LANNION

UNIVERSITÉ DE RENNES 1

Développement d'un ordonnanceur spatio-temporel pour une plateforme reconfigurable dynamiquement

Development of a space-time scheduler for a dynamically reconfigurable platform

Tuteur : M. Daniel Chillet

Equipe de recherche INRIA - ENSSAT

6 Rue Kerampont

22 300 LANNION

Alexis Pasturel– EII 3

27 mars 2011

Table des matières

Table des figures

Les mots marqués d'une * dans la suite du document sont répertoriés ici.

Seule la page de la première occurence est notée.

Glossaire

Bitstream Fichier contenant toutes les informations nécessaires à la programmation du FPGA, pour une petite zone ou toute la surface programmable. 20

DSP Digital Signal Processor, processeur spécialisé dans le traitement de signal. 8

FPGA Field-Programmable Gate Array, circuit électronique reprogrammable. 8

Hopfield Physicien américain, connu pour son modèle de Hopfield, détaillé dans le document. 9

Implémentation Mise en pratique sur un circuit réel d'un système théorique. 9

Instance Une tâche est divisée en plusieurs parties, appelées instances, qui correspondent aux différentes zones matérielles pré-définies utilisées pour exécuter la tâche. 7

MicroBlaze Microprocesseur "virtuel", spécifique aux FPGA de la marque Xilinx et synthétisé ensuite en entité matérielle. 12

Ordonnancer Fait de donner un ordre d'exécution des tâches d'une application. 7

RANN Reconfigurable Artifical Neural Network, gère l'aspect temporel de l'ordonnancement. 11

Réseau de neurones Un neurone est une entité reliée à N autres, l'ensemble constituant un réseau. 9

Synthèse (avec placement et routage) Passage du système, codé dans un code haut niveau par le développeur, à un fichier décrivant les ressources utilisées pour la cible FPGA choisie, ainsi que le placement de ces ressources et les liens entre elles. 11

VHDL Very-high-speed integrated circuit Hardware Description Language, language de description matériel de haut niveau. 12

Abstract

This project has been realised within the research INRIA team CAIRN at the ENSSAT in Lannion. This team is specialised on the hardware system conception for reconfigurable plateform.

A space-time scheduler is necessary to optimize the execution of an application on a reconfigurable plateform. Indeed, during execution process, several tasks can be executed on a reconfigurable execution resource, and the temporal and the spatial organization must be defined. In order to optimize time and resources, a space-time scheluder is necessary for choose which instance will be used and where it will be implemented in the circuit at the next step of the execution.

This project consists on implementing a neural network architecture based on Hopfield model. Some publications and research works on this subject were made by Messrs Daniel Chillet, Sébastien Pillement, Olivier Sentieys and Antoine Eiche. With this project there will be some pratical results, on used area and maximal frequency of the chosen architecture.

This document presents the subject, the implementation choices, the work methology, the results obtained and follow-up project.

Introduction

Le projet est réalisé dans le cadre des projets technologiques à l'ENSSAT, au sein de l'équipe de recherches CAIRN, spécialisée dans la conception de systèmes matériels pour des plateformes reconfigurables, et particulièrement reconfigurables dynamiquement.

Sur ce type de plateformes, il est nécessaire d'ordonnancer* les tâches - comme dans n'importe quel système. Habituellement, l'ordonnancement est temporel, mais dans le cas du reconfigurable dynamiquement, cela suppose qu'il faut ordonnancer le placement des tâches à exécuter, pour s'assurer que deux tâches ne soient pas au même endroit au même moment. On parle alors dans ce cas d'ordonnancement spatio-temporel.

Ce projet concerne l'implémentation matérielle d'un modèle d'architecture développé pour l'ordonnancement spatio-temporel. Ce type d'ordonnancement sur une plateforme reconfigurable est important, car il permet d'optimiser l'exécution des instances* configurées sur la plateforme à un intstant t.

En effet, l'ordonnanceur pourra définir quelles tâches devront être exécutées, et donc il définira quelles instances - qui correspondent à ces tâches - devront être configurées, au prochain cycle d'exécution.

Ainsi, l'implémentation de ce modèle permettra d'obtenir des résultats pratiques à mettre en regard des résultats théoriques obtenus lors de recherches antérieures.

Ce document présente le sujet, la méthodologie de travail choisie, les résultats obtenus et les suites à donner au projet.

1 Description du sujet

1.1 Entreprise

L'équipe de recherche CAIRN est spécialisée dans la conception de systèmes matériels pour des plateformes reconfigurables. Elle y développe des modèles d'architectures et des méthodologies de conception.

L'équipe est constituée d'une cinquantaine de personnes dont une quinzaine d'enseignants chercheurs et d'ingénieurs, ainsi que d'une quarantaine de doctorants et d'ingénieurs.

Source : http ://cairn.enssat.fr/

1.2 Contexte

Ce projet s'inscrit dans un travail de recherche engagé par messieurs Daniel Chillet, Sébastien Pillement, Olivier Sentiyes, et Antoine Eiche, respectivement chercheurs et doctorant dans l'équipe de recherche CAIRN.

1.3 Problématique

Le problème global est l'ordonnancement spatio-temporel de tâches pour une plateforme reconfigurable dynamiquement. La plateforme ciblée est donc un FPGA* .
L'aspect reconfigurable est un but futur pour ce projet de recherche, mais le problème posé dans le projet technologique est concentré sur l'aspect ordonnancement. Une architecture reconfigurable peut être modélisée comme sur la figure 1. De plus, la particularité du projet consiste dans le fait que la zone reconfigurable est considérée hétérogène : toutes les zones ne sont pas identiques, certaines peuvent par exemple contenir des blocs de calcul DSP* .

Les travaux de recherches engagés en 2008 ont abouti à des résultats théoriques concernant le problème de l'ordonnancement temporel, tandis que le travail d'Antoine Eiche a permis de définir au sens mathématique le problème de l'ordonnancement spatial. Dans

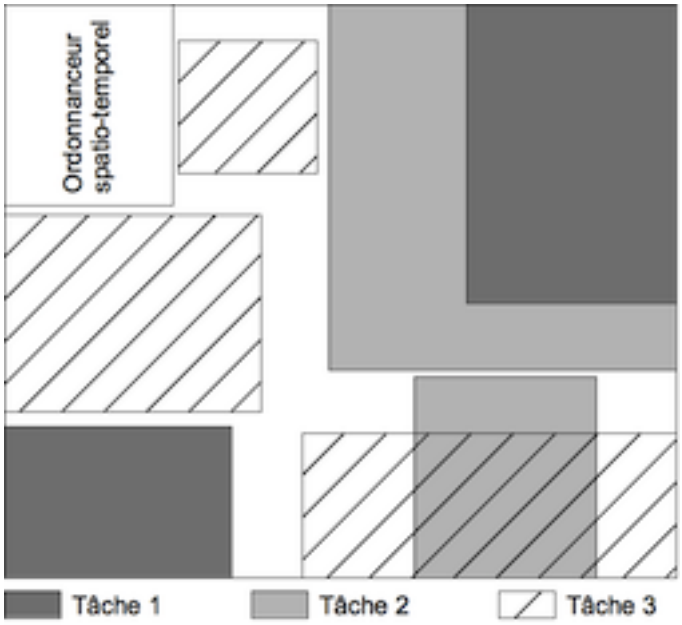

FIGURE *1 – Exemple d'architecture reconfigurable. Dans cette architecture, on remarque la présence de trois tâches, ainsi que les dépendances en ressources entre elles. En effet, les tâches qui se chevauchent ne peuvent être exécutées au même instant. De plus, dans cet exemple on remarque que la surface est truffée de zones non allouées et qui seront difficilement utilisables pour d'autres tâches. Dans l'aspect spatial de l'ordonnancement, le placement des instances de tâches est un paramètre à prendre en considération.*

les deux cas, le concept mathématique utilisé est un réseau de neurones* suivant le modèle définit par Hopfield* .

Dans cette modélisation, les neurones sont tous liés les uns aux autres par des arcs pondérés (cf figure 2). Chaque neurone reçoit tour à tour de l'énergie en entrée. Le choix du neurone qui reçoit l'énergie - c'est le neurone à évaluer - à un instant t dans le réseau est aléatoire. Le neurone peut avoir deux états en sortie : 0 ou 1. Son état de sortie est calculé en comparant l'énergie en entrée avec une somme pondérée (valeurs sur les arcs) des états de sortie des autres neurones. Une fois que les sorties des neurones sont toutes calculées et qu'elles sont stabilisées, on dit que le réseau a convergé.

Le problème posé dans ce projet est l'implémentation* matérielle des différents travaux réalisés, sur un circuit FPGA, dans le but d'obtenir des résultats pratiques en terme

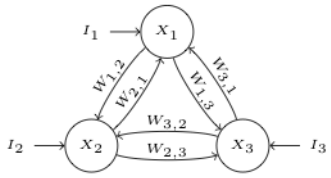

FIGURE *2 – Réseau de 3 neurones, selon Hopfield. On remarque les pondérations W sur les arcs, et l'énergie I en entrée.*

de temps de traitement et de surface utilisée pour la mise en oeuvre du système.

L'implémentation du modèle de Hopfield, tel qu'il est présenté, est très gourmande en ressources matérielles car elle demande un grand nombre d'opérations de multiplication, d'accumulation et de comparaison à chaque évaluation. Or, on cherche à implémenter un ordonnanceur, c'est-à-dire un petit bloc de contrôle d'une application. Il est donc nécessaire que ce bloc prenne le moins de ressources physiques, pour maximiser les ressources disponibles pour l'application.

Le réseau de neurones modélise l'application que l'on souhaite ordonnancer. L'application en question est divisée en plusieurs tâches, modélisées par un ou plusieurs neurones. De plus, il n'est pas possible d'instancier toutes les tâches au même moment, pour les raisons suivantes :

- Dépendances entre tâches : certaines tâches ne peuvent être exécutées avant que d'autres ne soient terminées, car elles ont besoin des résultats calculés par ces tâches (dépendances de données).
- Les ressources matérielles : la surface totale du FPGA peut ne pas être suffisante pour implémenter toutes les tâches s'il s'agit d'une grosse application.

Le rôle de l'ordonnanceur est donc de définir quelles tâches seront implémentées à un instant t. Une fois ces tâches exécutées, il sera possible de reconfigurer la surface utilisée pour permettre l'exécution d'autres tâches qui n'ont pas encore été ordonnancées.

2 Méthodologie de travail

Ce projet est une suite par rapport à des travaux déjà effectués précédemment.

L'aspect temporel a déjà fait l'objet d'une publication[1] ainsi qu'un début de travail sur l'implémentation, par M. Sébastien Pillement, puis par moi-même dans le cadre d'un stage. L'aspect spatial est resté au stade de l'étude théorique, et a fait l'objet d'un article [2]. De plus, un article présenté à la conférence SYMPA à Saint-Malo courant mai décrit les résultats obtenus durant ce projet sur l'implémentation du RANN[4].

Le problème est décomposé en trois parties :

1. Aspect temporel : Implémentation du réseau RANN*
2. Aspect spatial : Définition d'une architecture matérielle viable
3. Aspect spatial : Implémentation de l'architecture dégagée

L'aspect temporel consiste en la terminaison du travail entrepris durant le stage. L'aspect spatial nécessite avant tout une étude du modèle proposé par M. Antoine Eiche dans [2]. Enfin, le travail réalisé sur l'aspect temporel a été mis à contribution pour développer le réseau à implémenter. Les résultats attendus à la fin du projet sont ceux obtenus en fin de chaque partie :

- Partie 1 : Le résultat de synthèse* : on retrouve dans le rapport de synthèse les données concernant la fréquence maximale d'utilisation et la surface - les ressources matérielles - utilisée pour implémenter le système modélisé.
- Partie 2 : Le modèle de réseau de neurones implémentable, et minimisant les ressources matérielles ou la fréquence d'utilisation.
- Partie 3 : Equivalent à la partie 1. Une architecture basique d'ordonnanceur spatial, similaire au RANN a été dégagée.

3 Travail réalisé

Un générateur écrit en php permet de créer facilement le système complet d'ordonnanceur à partir des caractéristiques du réseau souhaité : on peut citer par exemple les dépendances, les surfaces ou les temps de cycles relatifs à chacune des tâches.

Toutes les architectures développées ont demandé la modification du générateur pour s'adapter aux modifications apportées, aux nouveaux arguments ou aux retraits d'informations car inutiles.

3.1 Ordonnancement temporel

Le réseau de neurones est implémenté tel que défini dans [1]. Chaque tâche est représentée par un neurone dans le réseau (cf Fig. 3).

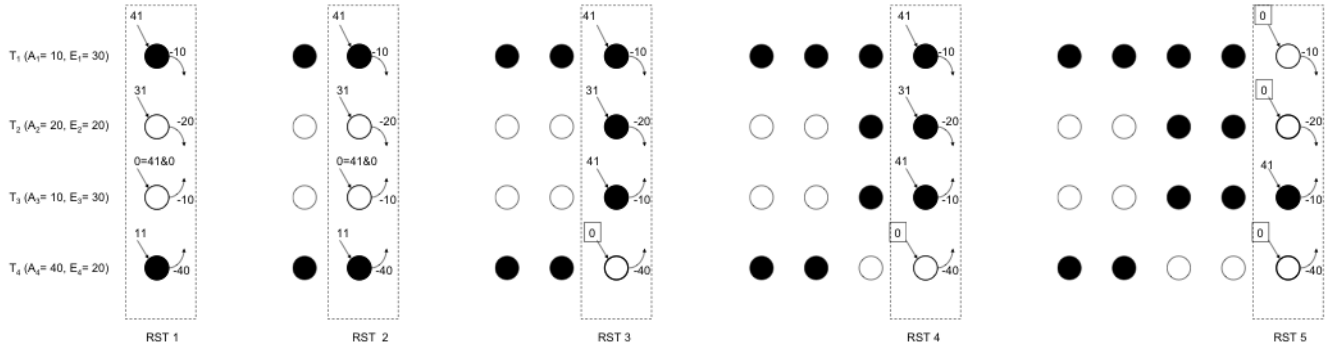

FIGURE *3 – Exemple d'évolution du réseau de neurones RANN pour plusieurs cycles d'ordonnancement (RST). Dans cet exemple, quatre tâches sont considérées. Chaque cycle d'ordonnancement s'assure que la surface occupée par les tâches est inférieure ou égale à la surface totale disponible dans la zone reconfigurable. Le contrôle du réseau permet de contrôler la non préemption des tâches ainsi que les dépendances entre les tâches. On peut remarquer le forçage des entrées à 0 pour les tâches ayant terminé leur ordonnancement ou dépendant d'autres tâches.*

Celui-ci a été développé en VHDL* par M. Sébastien Pillement, et a fait l'objet de simulations. Un contrôleur y a été ajouté - il permet de calculer le nombre de cycles d'ordonnancement exécutés pour chaque tâche, et désactive chaque neurone une fois la

tâche associée terminée -, ainsi qu'un microprocesseur MicroBlaze* , gérant l'aspect "choix aléatoire du neurone".

Dans un premier temps, le contrôleur était intégré au MicroBlaze. Afin de limiter la dépendance du système développé au processeur, pour permettre une implémentation sur n'importe quel FPGA, il a été décidé de l'en extraire. Dans les deux cas, on peut représenter le système comme sur la figure 4.

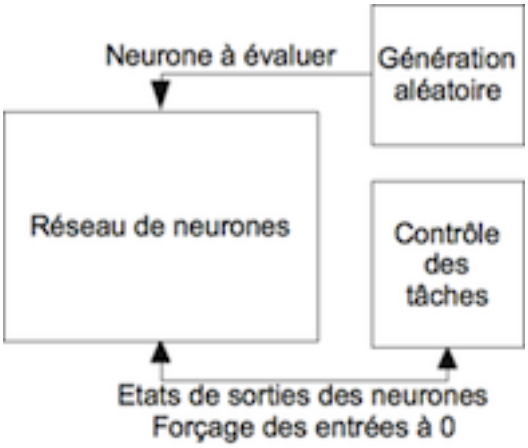

FIGURE *4 – Architecture générique du système développé. Le réseau de neurones est de type RANN. La génération aléatoire et - ou non suivant la version - le contrôleur de tâches sont gérés par le MicroBlaze. Le réseau est évalué avec la suite générée aléatoirement. Le contrôleur vérifie après chaque cycle de convergence s'il doit forcer des entrées à 0.*

La description ci-après décrit la dernière version de l'architecture développée (cf Fig. 5). Les deux autres seront détaillées par ailleurs.

3.1.1 Réseau de neurones et neurones

Le réseau en lui-même n'a pas de spécificité particulière. Il reçoit les signaux de commandes du contrôleur et de la machine d'états, qui sont renvoyés localement sur chaque neurone. Il redirige en sortie l'état des neurones.

En ce qui concerne les neurones, leur activation dépend de l'état des autres neurones. Un neurone sera actif en fonction du résultat d'une longue équation dont chaque membre

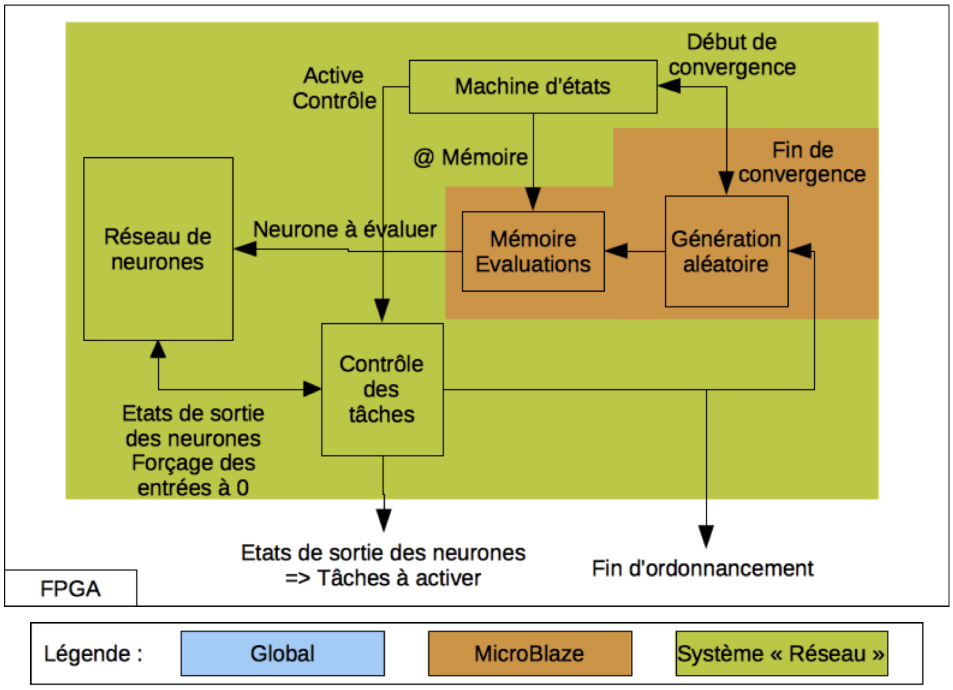

FIGURE 5 – *Architecture de l'ordonnanceur temporel développé, v3*

est une combinaison des états des autres neurones. C'est par ailleurs un problème qui sera développé par la suite.

On gère dans l'équation les dépendances de surface : connaissant la surface totale du FPGA (sans se soucier de son hétérogénéité) et la taille de chacune des tâches, on peut en déduire les tâches instanciables au même instant. L'équation du neurone X liste alors les combinaisons de neurones (tâches) dans lesquelles le neurone X est actif. Les cas où, de par sa surface la tâche ne peut pas être placée, sont alors gérés en rendant le neurone inactif.

3.1.2 Contrôleur

Le contrôleur du RANN prend en entrée le résultat de convergence du réseau. Au départ, il connait le nombre de cycles d'ordonnancement pour chaque tâche. Pour chaque neurone actif, il va décompter un cycle d'ordonnancement. Pour simplifier la comparaison, on se sert du bit de signe : si le nombre de cycle d'ordonnancements est négatif (égal à -1 en réalité), alors on considère que la tâche a terminé son exécution.

De plus, une fois une tâche terminée, le contrôleur va positionner une de ses sorties à 1. Ainsi, il bloquera l'activation du neurone associé à la sortie. Si on ne prend pas cette précaution, l'ordonnanceur va penser que la tâche (le neurone) peut être ordonnancée à ce moment là. Or, en considérant les dépendances liées à la tâche, en plus de réexécuter une tâche terminée (et donc fausser les résultats de calculs), on va empêcher l'exécution de tâches qui doivent réellement l'être.

3.1.3 Machine d'états

La machine d'états, qui peut être associée au contrôle dans les schémas blocs, gère l'interface entre le système "ordonnanceur temporel" et les éléments extérieurs, ainsi que le séquencement des tâches que doit effectuer l'ordonnanceur pour s'exécuter correctement. Elle n'est connectée visuellement qu'au réseau, mais un signal d'activation est relié au contrôleur. De plus, c'est elle qui adresse la mémoire utilisée pour stocker l'ordre dans lequel les neurones seront évalués.

Le séquencement est le suivant :

1. Init_Random : dans cet état, le périphérique en charge de calculer l'ordre dans lequel seront évalués les neurones pour la convergence courante est actif ; on attend qu'il ait terminé pour poursuivre, un bit est utilisé à cette fin sur le port d'entrée. Il passe à 1 pour signifier la fin de calcul.

2. Init_Eval : on démarre l'évaluation séquentielle des neurones du réseau.

3. EvalX : correspond aux différentes évaluations. Le paramètre important dans cet état est l'adressage de la mémoire qui change à chaque évaluation.

4. Control : c'est à cet instant que le contrôleur effectue le travail décrit dans la section 3.1.2. Le signal sur le port de sortie de fin de convergence passe à 1 également. Ce signal s'adresse au générateur pseudo-aléatoire utilisé dans l'état Init_Random.

5. Attente : cet état est important car il faut attendre que le générateur pseudo-aléatoire ait bien pris en compte la fin de la convergence et qu'il reprenne la main.

L'information est connue quand le bit sur le port d'entrée (utilisé également dans l'état Init_Random) passe à 0.

3.1.4 Générateur aléatoire, évaluations de neurones

La génération aléatoire nécessaire pour effectuer le choix (l'évaluation) du neurone lors du cycle de convergence a été réalisée avec le microprocesseur MicroBlaze interne au FPGA. L'aspect aléatoire n'est pas un objectif recherché dans ce projet, c'est pourquoi il a été décidé d'utiliser le microprocesseur, programmé en C, pour faciliter la tâche.

Le principe de fonctionnement est le suivant : à chaque début de cycle de convergence (état Init_Random de la machine d'états), le MicroBlaze tire les neurones dans un ordre aléatoire en utilisant la fonction random propre au langage C. Il enregistre la liste ensuite dans la mémoire BRAM (32 bits) sous la forme suivante : chaque ligne de 32 bits contient l'évaluation d'un neurone. Ce neurone est désigné par le seul bit à 1 de la valeur enregistrée. Ce bit est celui qui est directement relié sur le signal d'évaluation du neurone.

3.1.5 Problèmes rencontrés

- Temps de synthèse : ISE, l'outil Xilinx utilisé pour la synthèse, est très long pour faire la synthèse lorsque le réseau est très grand (environ trois jours pour quinze neurones par exemple). Ceci est du à la complexité de l'équation interne au neurone : l'outil va la décomposer en arbre pour essayer de la simplifier. Or si on a un très grand nombre de neurones et un certain nombre de possibilités, l'arbre est trop grand pour l'outil et il fait alors un segmentation fault lors du parcours (dépassement de mémoire).

  La solution pour pallier à ce problème est l'utilisation d'un autre outil de synthèse, Synplify. Le seul problème rencontré avec ce nouvel outil est l'intégration du microprocesseur MicroBlaze dans la boucle de synthèse. En effet, on ne peut pas effectuer la synthèse comme sous ISE en insérant tous les blocs. Il faut effectuer la synthèse du MicroBlaze sous XPS (Xilinx Plateform Studio) auparavant.

Cependant, le problème a été déplacé car il n'a pas été possible d'augmenter de façon significative le nombre de neurones. La synthèse même sous Synplify n'a pas abouti avec 25 neurones.

- Bonne gestion du contrôleur : la sortie du contrôleur du MicroBlaze n'a pas posé de problème côté MicroBlaze, mais la gestion générale du système s'est un peu complexifiée. Le MicroBlaze est lent, chaque instruction demande un nombre important de cycles d'horloge. Le réseau et le contrôleur étant à la fréquence d'horloge, on effectue plusieurs calculs en parallèle d'un seul sur le MicroBlaze. C'est donc très sensible au cycle d'horloge : un décalage sur un signal entraîne une incohérence des résultats. C'est ce problème qui a conduit à l'ajout de l'état Attente dans la machine d'états, car le temps que le bit d'entrée passe à 0 on redémarrait un cycle de convergence.

- Enfin, la mise en œuvre du MicroBlaze pour faciliter la tâche l'a en réalité compliqué d'une certaine façon : il n'est pas aisé d'implémenter le microprocesseur sans quelques connaissances particulières. Beaucoup de temps, de recherche de solution et en synthèse, a été perdu à cause d'une mauvaise connexion.

  Pour assurer une bonne synchronisation du MicroBlaze avec le reste du système, il est nécessaire de faire entrer l'horloge générale du circuit dans le MicroBlaze et d'utiliser une horloge générée par ce dernier pour tous les périphériques. Le même problème se pose pour le signal de reset.

  De plus, la mise en place d'une BRAM s'effectue directement dans le fichier système du projet. On doit y décrire les différents ports à utiliser (BRAM_ADDR_B par exemple).

3.1.6 Résultats de synthèses et évolutions de l'architecture

La synthèse s'est effectuée sur une plateforme d'évaluation Xilinx ML507, possédant un FPGA Xilinx Virtex 5 de 44 800 slices registers.

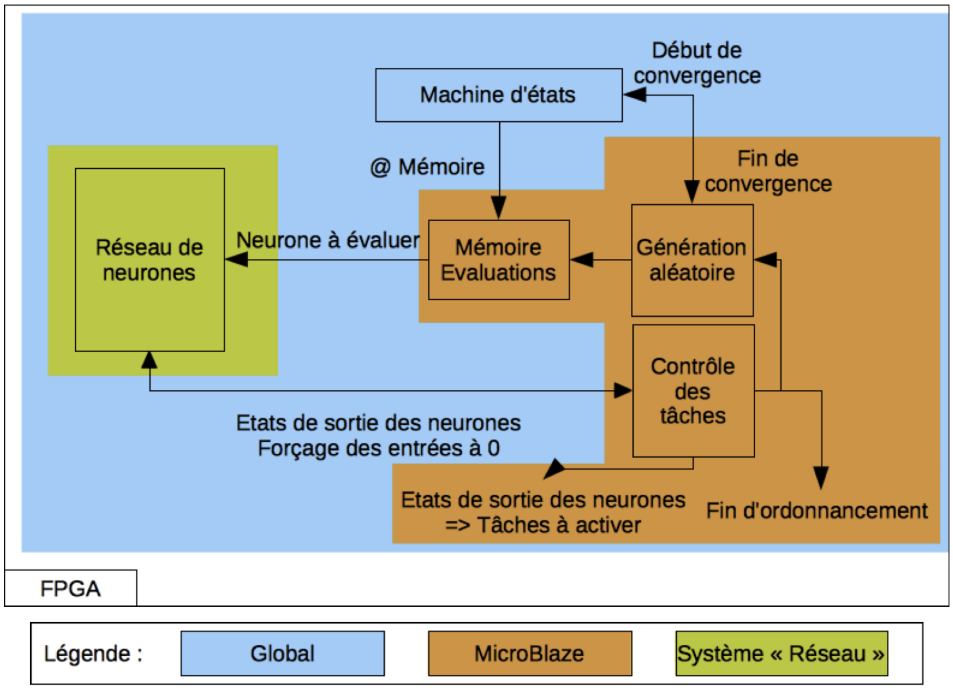

FIGURE 6 – *Architecture de l'ordonnanceur temporel développé, v1*

Une première synthèse a été effectuée avec le contrôleur en interne du MicroBlaze (cf Fig. 6), le tout toujours piloté par une machine d'états externe. Dans cette version, le contrôleur est géré par le MicroBlaze. Les résultats sont donnés pour des réseaux allant de quatre à quinze neurones. Les synthèses ont été effectuées avec l'outil ISE de Xilinx, et avec deux types d'optimisation : surface ou vitesse.

Globalement, les résultats sont équivalents entre les deux optimisations. Le réseau seul peut tourner à une fréquence allant de 775 MHz pour quatre neurones à 280 MHz pour quinze neurones. L'ajout de neurones augmente donc le chemin critique du circuit. En terme d'espace, le réseau ne prend qu'à peine 1% du FPGA, c'est donc un bon point puisqu'on souhaite minimiser l'espace requis. En insérant le MicroBlaze dans la synthèse, le réseau est presque transparent, le processeur est relativement gourmand en ressources (2000 slices contre 400). Néanmoins, on reste aux alentours de 4 à 5% de surface utilisée. La fréquence maximale est fixe, à 186 MHz, quelque soit le nombre de neurones. C'est donc le MicroBlaze qui fixe le temps de traitement, pour ces réseaux.

Le problème de cette première version est sa dépendance avec le MicroBlaze. Ce

dernier ayant été inséré pour simplifier la partie "génération aléatoire" qui n'est pas l'objet de l'étude, il ne doit pas avoir une forte influence sur les résultats obtenus. Or, le contrôle est un élément majeur du système, et il se doit d'être intégré aux résultats. Il est donc nécessaire de l'extraire du MicroBlaze. Ceci a donné lieu à la version 2 de l'architecture (cf Fig. 7). Aucun résultat n'a été tiré de cette version, la suivante est arrivée très vite après.

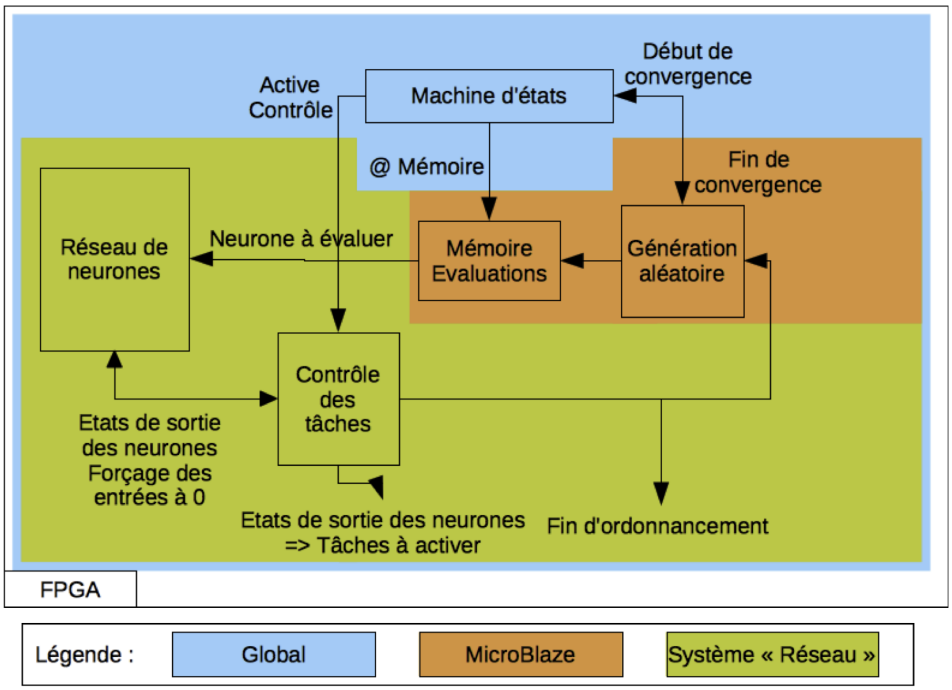

FIGURE 7 – *Architecture de l'ordonnanceur temporel développé, v2*

Un nouveau problème, légèrement similaire au précédent, s'est alors posé : la machine d'états doit elle aussi faire parti du résultat, mais il faut pouvoir ne pas synthétiser le MicroBlaze. Pour cela, il faut donc insérer la machine d'états avec le réseau et donc "décrocher" le MicroBlaze de tout le reste. Il est toujours nécessaire pour une implémentation fonctionnelle, mais il est possible d'obtenir des résultats sur ce qui nous intéresse ici. Ceci a donc été la troisième et dernière version développée dans ce projet.

Pour cette version (cf Fig. 5), des résultats de synthèses en terme de surface et de fréquence d'utilisation ont été obtenus pour des réseaux allant de quatre à vingt neurones. Les synthèses ont été effectuées avec deux types de réseaux : avec ou sans dépendances. Une dépendance est donnée ici comme une dépendance de données : une tâche X ne

pourra pas être exécutée avant une tâche Y.

On peut faire des remarques similaires à la première version concernant les résultats obtenus avec les deux types de réseau (cf Fig. 8 et 9). Le réseau seul peut tourner à une fréquence allant de 600 MHz pour quatre neurones à 200 MHz pour vingt neurones. On remarque que les dépendances n'ont pas d'effet particulier sur la vitesse de fonctionnement.

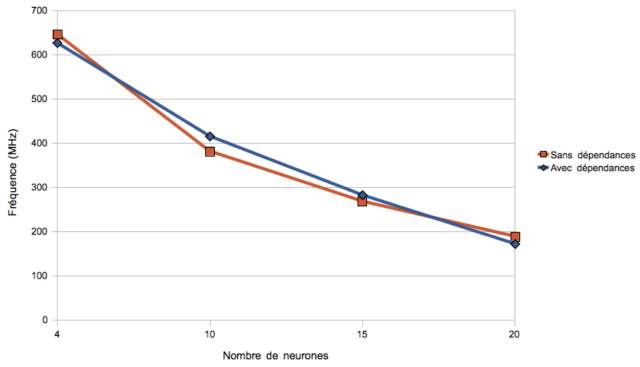

FIGURE *8 – Résultats de synthèse du RANN en terme de fréquence d'utilisation, avec ou sans dépendances*

En terme d'espace, le réseau ne prend qu'à peine 2,25% du FPGA avec vingt neurones, on tient toujours l'objectif d'utiliser le moins d'espace possible. Globalement les dépendances ont tendance à faire chuter le nombre de slices utilisés. Cependant le réseau généré pour quinze neurones a inversé la tendance. Ceci peut être du à la spécificité du réseau utilisé.

Il faut toutefois noter que ces résultats n'ont été obtenus que pour des réseaux bien particuliers, notamment ceux possédant des dépendances. Un même nombre de neurones avec des dépendances différentes donneraient probablement d'autres résultats. Ce sont donc des ordres d'idées que nous pouvons donner ici et non pas des vérités vraies.

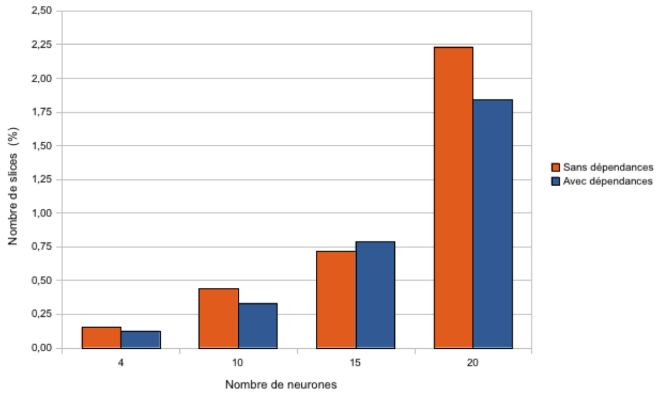

FIGURE *9 – Résultats de synthèse du RANN en terme de surface utilisée, avec ou sans dépendances*

3.2 Ordonnancement spatial

3.2.1 Définition du problème à mettre en œuvre

L'ordonnancement spatial a pour objectif de définir à quel endroit les tâches à exécuter doivent être placées sur le FPGA. Il intervient donc logiquement à la suite de l'ordonnancement temporel. Une fois les tâches à exécuter connues, il faut alors choisir une position. En reconfiguration dynamique, il est nécessaire de connaître à l'avance ce qu'on souhaite placer à quel endroit. Si on souhaite placer une tâche X à un endroit A ou un endroit B, il faudra donc donner au système au départ les bitstreams* correspondant à ces deux placements. Chaque bitstream correspond à ce qu'on appellera par la suite à une instance : une position possible de la tâche sur le FPGA.

On ne pourra placer une instance à un endroit que si aucune autre instance n'utilise déjà la zone matérielle. Ceci crée donc des dépendances. L'objectif étant de pouvoir placer toutes les tâches à exécuter à un instant t, il est donc nécessaire d'avoir un certain nombre d'instances pour chaque tâche, chaque instance plaçant la tâche à un endroit différent du FPGA.

On modélise chaque instance avec un neurone, sur le modèle de Hopfield, comme pour l'ordonnancement temporel. Néanmoins, les neurones ont une relation différente entre eux que dans le cas du temporel. En effet, les pondérations sur les arcs (pour les dépendances) vont dépendre de deux paramètres : si le neurone correspond à une instance

de la même tâche ou d'une tâche différente, et s'il existe une dépendance sur le placement (recouvrement, partage de ressources).

Pour une combinaison de tâches données, ces paramètres vont influer sur la convergence du réseau pour permettre une configuration optimale, notamment en utilisation des ressources du FPGA. Ceci a fait l'objet de l'article [2]. L'ordonnanceur temporel a donc pour objectif concret de choisir quelle(s) instance(s) il faut placer, en fonction des tâches que l'on souhaite exécuter.

3.2.2 Architecture développée

L'architecture développée lors de ce projet est basée sur le modèle du RANN. On retrouve le réseau de neurones, la machine d'états et le MicroBlaze, avec les mêmes rôles. Dans l'ordonnancement spatial, il n'est pas utile de savoir si une tâche a terminé son exécution ou non, il n'y a donc pas de contrôleur en tant que tel. Il reçoit en entrée le résultat de convergence du RANN, il est alors possible de désactiver (forcer la sortie à 0) les neurones correspondant aux instances des tâches à ne pas placer.

L'aspect "placement optimal" n'a pas pu être développé, nous y reviendrons par la suite. Les neurones sont tous égaux entre eux, à l'image du réseau utilisé pour le RANN. Le principe d'activation est le même : une équation logique. Cependant, cette fois l'équation est très simple, car on considère qu'un neurone ne pourra être actif que si l'instance qu'il représente peut être placée par rapport aux autres instances dont il dépend. Si on dispose de N neurones, l'équation n'aura donc au maximum que $N - 1$ éléments.

Le MicroBlaze est toujours utilisé pour effectuer une génération aléatoire sur les évaluations. Sa complexité est plus élevée que pour l'ordonnancement temporel, car il ne doit générer une suite aléatoire que pour les instances des tâches à exécuter. Il a donc besoin du résultat du RANN en entrée pour faire un tri avant la génération.

La machine d'états fonctionne de façon similaire à celle de l'ordonnancement temporel. Elle a été modifiée lors de l'élaboration de l'ordonnancement spatial. Plus précisément, elle a été généralisée. Celle qui est présente dans l'architecture de l'ordonnancemeur temporel

possède un état pour l'évaluation de chaque neurone. Or pour l'ordonnancement spatial, on ne sait pas a priori combien d'évaluations seront effectuées, leur nombre peut changer à chaque cycle de convergence. On a donc un seul état dans la machine d'états pour toutes les évaluations effectuées sur le réseau.

3.2.3 Problèmes rencontrés

La mise en œuvre étant similaire, voire simplifiée, à l'ordonnanceur temporel, aucun problème majeur ne s'est posé lors du développement. C'est lors de l'implémentation qu'un soucis est apparu : le nombre de broches d'interface entre le MicroBlaze et le réseau de neurones s'est avéré trop petit. En effet, la sortie de la mémoire BRAM utilisée est sur 32 bits, ce qui fait donc 32 neurones (1 bit d'évaluation par neurone). Or, pour un test "réel", nous avons souhaité mettre en œuvre un application sur une grille 10x10, avec dix tâches et dix instances par tâche. Soit 100 neurones.

Pour pallier à ce problème, deux opérations sont nécessaires :

- Augmenter la taille de la BRAM : il est possible, en associant deux BRAM 64 bits de disposer d'une mémoire sur 128 bits. Une BRAM 64 bits a été testée seule, sans connexion avec le réseau. Le problème que cela pose, en dehors d'user de beaucoup de ressources qui ne sont disponibles qu'en petit nombre, c'est que la lecture et l'écriture ne sont possibles que sur 32 bits. Ceci complexifie sérieusement la façon dont on interagit, que ce soit côté MicroBlaze que côté réseau de neurones. En effet, pour évaluer un neurone, il sera nécessaire d'effectuer quatre lectures pour être sûr d'avoir lu la partie de 32 bits qui contient le seul bit à 1. L'assemblage de deux BRAM 64 bits n'a pas été effectué, la mise en œuvre d'une seule BRAM 64 bits ayant pris plus de temps que prévu. Un travail sur le contrôleur de BRAM est à effectuer pour réussir à assembler deux mémoires ensembles.

- Modifier les interfaces : les ports I/O sont sur 32 bits et ne peuvent pas être agrandis. Il faut donc en prévoir suffisamment (théoriquement quatre si on utilise une BRAM 128 bits), et pouvoir gérer ensuite les lectures côté MicroBlaze.

Aucune implémentation "corrigée" n'a été réalisée.

4 Ordonnancement spatio-temporel

Aucune implémentation de l'ordonnanceur complet n'a été réalisée, néanmoins l'assemblage des deux ordonnanceurs a fait l'objet de réflexions.

Un des points soulevés est le fait que l'ordonnanceur temporel doit tenir compte du résultat de l'ordonnanceur spatial. En effet, il peut arriver que ce dernier ne puisse pas placer toutes les tâches demandées par le RANN. Il faut alors qu'au moment du contrôle, l'ordonnanceur temporel ne décompte pas de temps de cycle pour les tâches qui n'ont pas pu être placées. Le fonctionnement de la machine d'état dans l'ordonnanceur temporel doit donc tenir compte de cet aspect.

De plus, les deux ordonnanceurs utilisant un MicroBlaze, il semble logique de rassembler les deux, ce qui donnerait une architecture du type de la figure 10. Cela limite par ailleurs le nombre de signaux qui transitent entre les deux architectures, et cela peut améliorer les performances et la simplicité du code. En effet, les résultats d'un ordonnanceur sont immédiatement connus du suivant.

On peut éventuellement imaginer un fonctionnement pipeline des ordonnanceurs où l'ordonnancement temporel T est effectué pendant l'ordonnancement spatial T-1. Il est difficile d'aller plus loin en profondeur car si l'ordonnanceur spatial n'a pas pu atteindre son objectif, il faut pouvoir en tenir compte pour dévalider le résultat de l'ordonnancement temporel et permettre un ordonnancement et une exécution correcte de l'application.

5 Revue critique

Il est juste de se demander si une autre solution que celle présentée ici (par réseau de neurones) n'aurait pas pu être envisagée. On aurait tout à fait pu concevoir un système totalement software, ou mixte avec un MicroBlaze pour la partie software et quelques parties accélérées matériellement. Le principe de l'ordonnanceur étant de savoir ce que doit faire le système sur lequel il travaille avant qu'il n'ait terminé sa tâche, il faut qu'il

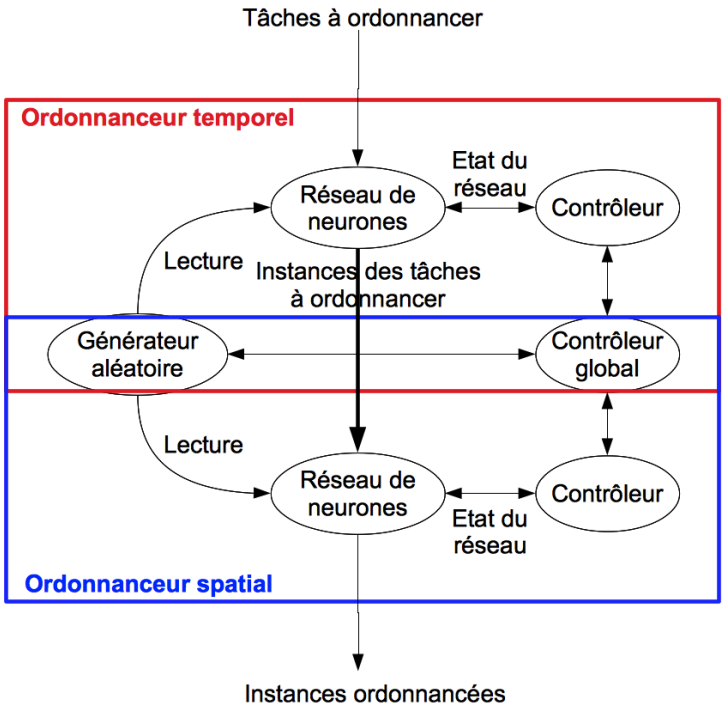

FIGURE *10 – Architecture de l'ordonnanceur spatio-temporel*

effectue tous ses calculs aussi vite que possible. C'est pourquoi une solution hardware est celle qui semble la plus judicieuse.

Néanmoins, comme il a été constaté sur les résultats de l'ordonnanceur temporel, les vitesses de fonctionnement descendent rapidement. Il est probable qu'arrivé à un certain nombre de neurones, une solution software serait plus rapide qu'une solution totalement hardware, avec cette architecture.

De plus, l'architecture en réseau de neurones possède un avantage de tolérance aux défaillances que d'autres systèmes n'ont pas. Ceci est détaillé dans [3]. L'ordonnanceur ne sera plus en mesure d'effectuer un ordonnancement correct si la partie contrôle vient à être défaillante. Mais s'il s'agit d'un neurone, soit il sera toujours actif, soit il ne le sera jamais, mais en tous les cas le reste du réseau pourra continuer à fonctionner, dans un mode dégradé. Quant au générateur aléatoire, s'il effectue des tirages non-aléatoires, cela n'aura qu'un impact sur la convergence dans le sens où un avantage sera donné à certains neurones pour s'activer. Cependant, s'il n'y a plus de génération, on rejoint le cas du contrôleur, l'ordonnancement même dégradé n'est plus possible.

6 Perspectives

Il pourrait être intéressant de généraliser de la même façon que dans l'ordonnanceur spatial, la machine d'états présente dans l'ordonnanceur temporel, pour diminuer sa taille.

L'utilisation du MicroBlaze a simplifié le problème de l'ordonnanceur temporel, mais il a bloqué le bon fonctionnement de l'ordonnanceur spatial. Il serait intéressant de s'en séparer et de se demander si l'aspect aléatoire est primordial ou non. La mise en œuvre de l'ordonnanceur spatial s'est trouvée stoppée pour une raison externe au coeur du projet. Peut-être qu'une solution moins aléatoire permettrait toutefois de pouvoir tester et surtout synthétiser des réseaux de plus grande taille. Sachant que la limitation due aux neurones de l'ordonnanceur temporel n'est plus du même ordre, il doit être possible d'en augmenter le nombre de façon significative.

Il véritable assemblage des deux ordonnanceurs ne peut être réalisé qu'après avoir trouvé une solution correcte, fonctionnelle et implémentable de l'ordonnanceur spatial.

Par ailleurs, un contrôleur général peut également être envisagé pour mettre fin à des cycles de calculs s'il se rend compte qu'une solution est trouvée. C'est assez restreint pour l'ordonnanceur temporel car il n'est pas possible de savoir a priori si une combinaison est encore possible. Néanmoins, le réseau pour le temporel est assez petit, le contrôleur concernait surtout l'ordonnanceur spatial. Si on sait que X tâches doivent être placées, on peut stopper les cycles d'évaluations dès que les X placements sont trouvés.

Par ailleurs, sauf cas de dépendances, en général les instances placées sont à la première apparition de chaque tâche dans la liste. Ceci signifie qu'avant d'avoir évalué la moitié, voir le quart des neurones, la solution est déjà trouvée. Il serait peut-être intéressant de voir s'il est possible de passer l'évaluation de certaines instances pour gagner du temps.

Références

[1] D. Chillet, S. Pillement, and O. Sentieys. *Algorithm-Architecture Matching for Signal and Image Processing, Springer*, chapter RANN : A Reconfigurable Artificial Neural Network Model for Task Scheduling on Reconfigurable System-on-Chip. Springer Verlag, 2010.

[2] A.Eiche, D. Chillet, S. Pillement, and O. Sentieys. Task placement for dynamic and partial reconfigurable region. In *Conference on Design and Architectures for Signal and Image Processing*, Edinburgh, October 2010.

[3] D. Chillet, A Eiche, S. Pillement, and O. Sentieys. Exploitation du concept de tolérance aux fautes des réseaux de neurones pour la résolution de problèmes d'optimisation. In *Soumis à Gretsi 2011*, 2011.

[4] A. Eiche, A. Pasturel, D. Chillet, S. Pillement, and O. Sentieys. Implémentation matérielle d'un réseau de neurones pour l'ordonnancement temporel de tâches sur architecture reconfigurable. In *RenPar'20 / SympA'14 / CFSE'8*, Saint-Malo, May 2011.