SETIT 2007
4rth International Conference: Sciences of Electronic, Technologies of Information and Telecommunications
March 25-29, 2007 – TUNISIA

Un « Système Multiprocesseur sur Puce » Flexible pour le Traitement d'Image et le Traitement de Signal

Majed Chatti[*,**], Fabrice Lemonnier[*], Claude Timsit[**] et Soraya Zertal[**]

[*]*Thales Research & Technology*

Route Départementale 128, Palaiseau (91767), France
majed.chatti@thalesgroup.com
fabrice.lemonnier@thalesgroup.com

[**]*Laboratoire PRISM*

Université de Versailles Saint Quentin en Yvelines 45, avenue des états unis, Versailles (78000), France
majed.chatti@prism.uvsq.fr
claude.timsit@prism.uvsq.fr
soraya.zertal@prism.uvsq.fr

Résume : Les besoins de l'industrie militaire et civile ne cessent d'évoluer. Aujourd'hui tous se tournent vers la mobilité et vers de nouvelles applications. Ces applications souvent de type traitements complexes sur flot de données continu nécessitent de plus en plus de puissance de calcul. L'industrie des microprocesseurs a su pendant un temps répondre aux exigences des utilisateurs en affichant des temps de réponse plus que satisfaisants sur ce type d'application. Cependant l'évolution de la puissance des microprocesseurs a souvent été synonyme d'augmentation de nombre de transistors et de fréquence de fonctionnement. Pour des contraintes purement physiques, ce type d'évolution arrive bientôt à ses limites. La solution consisterait peut-être à passer à une nouvelle génération de processeurs. Le MPSoC (Multi Processor System on Chip) parait être une voie prometteuse. Les MPSoCs, introduisent de nouvelles contraintes et de nouveaux défis à soulever, dus essentiellement à leur nature multi-cœurs. Dans ce nouveau défi que représentent ces nouvelles architectures multiprocesseurs et en se rattachant au contexte des nouvelles applications de traitement de flot continu de données, nous présentons un modèle de MPSoC pour traitement d'image et traitement de signal.
Mots clés : Traitement d'image, Traitement de signal, MPSoC, STAP.

Introduction:

Aujourd'hui la mobilité est devenue le mot clef, l'attrait, la vocation de plusieurs nouvelles recherches initiées autant par le secteur public que militaire. Ce besoin de mobilité couvre des domaines aussi variés que la sécurité, la défense, le multimédia ou encore le transport ; d'où une nécessité grandissante d'augmenter les puissances de calcul des machines embarquées sans pour autant augmenter, voire même de diminuer prodigieusement, la consommation de ces machines. Ces nécessités viennent du fait que l'utilisateur final revendique de plus en plus le droit à une meilleure précision (augmenter le taux de fiabilité des Radars embarqués), à une meilleure qualité (imagerie, vidéo…) et à des temps d'attente sans cesse plus courts (performance et réactivité élevées). De plus, nous assistons à l'éclosion de nouvelles applications, telle que la reconnaissance de cibles sur image ou sur vidéo, dues principalement aux progrès réalisés en termes de puissance de calcul. Néanmoins, ces derniers restent insuffisants car ne garantissant pas toujours des contraintes temps réels, consommation d'énergie ou coût de développement et de production.

Les solutions matérielles usuelles pour traiter ces types d'application temps réels à flot continu de données sont longtemps restées de deux types.

- Une solution dédiée : consiste en la conception d'un circuit intégré spécifique ou **ASIC** (Application Specific Integrated Circuit) pour une ou plusieurs fonctionnalités bien déterminées. Ceci implique donc qu'il est inégalable en termes de performance et de consommation pour l'application ou la fonctionnalité pour laquelle il a été conçu. Cependant, le coût de développement d'un **ASIC** demeure très élevé. Ce coût de développement et de fabrication appelé **NRC** (Non Recurring Cost) ne peut être réduit qu'en multipliant le nombre de circuits produits. Or, ceci est difficilement envisageable pour les ASIC vu qu'ils sont à usage spécifique donc réduit. En raison de son coût, la solution **ASIC** n'est donc pas toujours envisageable.

- Une solution généraliste consiste en l'utilisation d'un processeur généraliste ou **GPP** (General Purpose Processor). Celui-ci est programmable donc, contrairement aux **ASICs**, est multi-applications. De plus, visant un marché plus ouvert, le **GPP** et par conséquent beaucoup moins onéreux. Cependant, ces processeurs généralistes, grands consommateurs d'énergie, n'offrent quasiment jamais les puissances de calcul nécessaires à une application à flot de données comme le traitement d'image ou de signal. Les **GPPs** n'offrent donc ni la puissance de calcul, ni la consommation réduite qui permettra de les utiliser de manière embarquée surtout dans le contexte des nouveaux besoins de l'industrie civile et militaire. Par ailleurs les DSPs (Digital Processing Processors) sont certainement des solutions intéressantes pour traiter les applications à flots de données. Cependant, ceux-ci étant, avant tout, des processeurs généralistes, présentent les mêmes limites que les GPPs. De plus les DSPs offrent aujourd'hui des puissances de calcul de l'ordre de 10GOPS théorique (le TMS320C64x de Texas Instrument) ce qui reste en deçà du contexte considéré.

Un autre point non moins important, est le fait que l'évolution de la technologie d'intégration et l'accroissement des fréquences des processeurs ont été longtemps les seules réponses à une augmentation sans cesse croissante des besoins des applications et des utilisateurs. Depuis quelques années déjà, d'autres solutions pour pallier ce problème de limite des performances des processeurs dû à des contraintes purement physiques sont à l'étude. L'une des voies les plus prometteuses consiste à multiplier le nombre d'unités de traitement sur une même puce d'où la naissance de ce qu'on appelle aujourd'hui les **MPSoCs** (Multi-Processors Systems on Chip).

Dans la suite de cet article, nous présenterons un modèle de multiprocesseur sur puce destiné autant au traitement de signal qu'au traitement d'image. Pour cela nous allons commencer par décrire le paysage des **MPSoCs** tel qu'il est aujourd'hui, nous présenterons ensuite notre vision de ce que devrait être une architecture **MPSoC** garantissant les contraintes décrites précédemment. Nous proposons par la suite une architecture **MPSoC** à laquelle nous joignons un exemple d'application de traitement de signal à savoir l'algorithme de traitement radar **STAP** ainsi que le placement de cet algorithme sur le **MPSoCs** proposé. Nous terminerons par une description des travaux futurs qui permettront de valider et d'améliorer le **MPSoC** proposé.

1. Multiprocesseur sur puce (MPSoC):

1.1. Etat de l'art :

Les multi-cœurs ou **MPSoCs** viennent comme une réponse aux limites des processeurs mono-cœur en puissance de calcul et en consommation d'énergie. En effet, l'augmentation des fréquences des processeurs ne pourra rester indéfiniment l'unique solution à une demande accrue de puissance de calcul, d'autant plus que l'augmentation de la fréquence entraîne non seulement une augmentation de la puissance de calcul, qui reste malgré tout en deçà des nécessités des nouvelles applications multimédias à flots de données, mais aussi particulièrement et inévitablement celle de la consommation. La fin de la loi de Moore[1] étant de plus en plus proche, on se tourne donc aujourd'hui vers d'autres solutions. Multiplier les processeurs sur une même puce semble être la voie prometteuse. A partir d'une étude faite sur quelques architectures **MPSoCs** [ARM 05] [CRA] [Hoe 05] [kah 05] [Phi 02] [ST 04] [TI 06], nous notons l'émergence de deux tendances :

- La première consiste à coupler un processeur généraliste (**GPP**) avec des accélérateurs dédiés aux traitements attribués à l'architecture. Ces accélérateurs peuvent être de types différents allant des accélérateurs graphiques (2D/3D, JPEG2000), audio (MP3) ou encore vidéo (MPEG 4), aux accélérateurs vectoriels (DSP). L'**OMAP** [TI 06] de Texas Instruments en est l'exemple, il comporte un processeur généraliste **ARM11** et trois types d'accélérateurs (2D/3D Graphic Accelerator, DSP et Imaging Video Accelerator).

- La deuxième tendance consiste à multiplier le nombre du même processeur généraliste sur une seule puce. Le **MPCore** de **ARM** [ARM 05] qui comporte quatre **ARM11** avec des mémoires distribuées en est un exemple

[1] Formulée par Gordon E. Moore en 1965, elle postule le doublement tous les deux ans des performances des circuits intégrés (mémoires et processeurs).

Hormis ces deux tendances, on distingue le processeur **Cell** (**IBM**, **Toshiba** et **Sony)** [Gsc 06] [Pen 05] [Hoe 05]. Celui-ci dispose d'une architecture hétérogène **(figure1)** qui comporte un processeur généraliste *PPE* (Power Processing Element) couplé avec huit accélérateurs vectoriels *SPEs* (Synergistic Processing Elements). Le *PPE* est un cœur **PowerPC** attribué essentiellement au contrôle mais qui peut aussi prendre en charge une partie du calcul ; Il dispose de deux niveaux de mémoire. Les *SPEs* représentent la principale nouveauté du Cell. Un *SPE* est composé de trois unités. Le *SPU* (Synergistic Processing Unit) représentant l'unité de calcul : il s'agit d'un processeur **SIMD** qui opère sur un vecteur de quatre éléments de 32 bits chacun. Le *LS* (Local Store) est un espace d'adressage local au *SPE*. Le *MFC* (Memory Flow Contrôle) est l'unité qui pilote les mouvements de données et l'accès au *LS*.

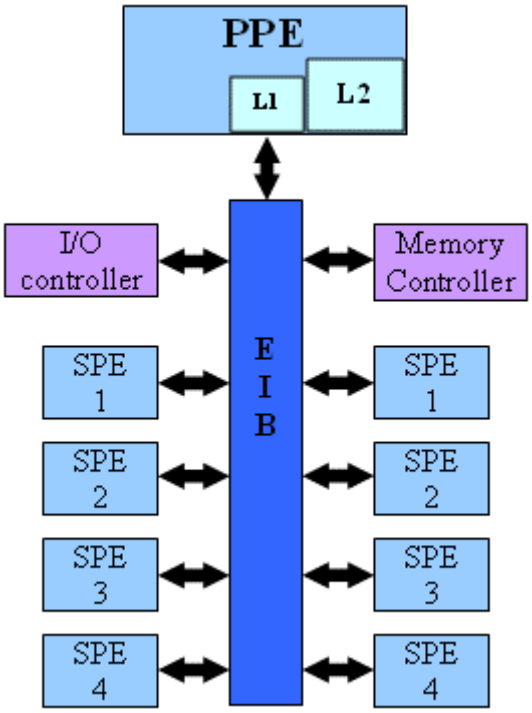

Figure 1. *Le processeur Cell.*

Outre le *PPE* et les *SPEs,* le Cell comporte un ***EIB*** (Element Interconnect Bus) ainsi que des unités d'accès à une mémoire externe et des unités d'entrées/Sorties.

1.2. Conclusion :

Les solutions citées précédemment sont certes intéressantes et répondent, plus ou moins efficacement, aux exigences des utilisateurs. Cependant, et sans rien ôter aux mérites de telles solutions, elles présentent toutes une ou plusieurs faiblesses ce qui nous a permis de les classer en deux catégories :

- Des solutions dédiées : ces solutions affichent des performances très intéressantes mais présentent des coûts élevés. De plus, les coûts de développement des puces sur silicium ne cessent d'augmenter du fait de la complexité accrue des techniques de fonderie ; cela étant malheureusement inévitable, la seule solution pour amortir le coût de développement et ainsi le prix d'achat d'un processeur reste la multiplication du nombre de puces créées et vendues. Les **MPSOCs** dédiées comme l'**OMAP** ciblent une tranche réduite du marché ne permettant pas d'arriver à produire un nombre suffisamment élevé pour amortir les coûts exorbitants de développement et de fonderie et de proposer ainsi aux utilisateurs un produit efficace à un coût raisonnable. La solution serait donc un processeur multi-applications qui ciblerait par conséquent une tranche plus large du marché des microprocesseurs et qui aurait comme conséquence la réduction de l'impact du **NRC**.

- Des solutions généralistes ou peut être "Trop" généralistes : en effet à l'opposé des solutions dédiées, performantes mais trop coûteuses, les autres solutions proposées consistent en des processeurs totalement programmables pouvant, éventuellement, cibler tous types d'applications ; Ceci permet sans aucun doute de réduire considérablement le coût de revient des puces mais engendre une perte considérable en rendement par unité surface : pour un même traitement on observe une augmentation de la consommation et du nombre de transistors non utilisés. Enfin le caractère **GPP** de ces solutions ajoute souvent une complexité à l'architecture ainsi résultante. Ces solutions, de part les caractéristiques citées précédemment, ne sont donc malheureusement pas toujours applicables à des applications embarquées à fort caractère traitement intensif. Le processeur **Cell**, quant à lui, outre le caractère généraliste, est de plus caractérisé par une consommation élevée, due principalement à une fréquence de fonctionnement élevée.

Il est donc impératif d'éviter de concevoir un processeur complètement dédié sans pour autant tomber dans le piège de ce que nous avons appelé le "Trop" généraliste. La solution que nous proposons est de cibler un domaine d'application. Nous nous sommes intéressés aux applications embarquées à flot continu de données et plus précisément aux applications de traitement d'image et de traitement de signal. Ces applications sont certes différentes mais présentent certaines similitudes (exemple : le caractère flot de données) qui peuvent être utilisées afin de développer un **MPSoCs** qui leur soit efficace. Cibler plusieurs types d'application permet d'élargir la portée du processeur produit et ainsi d'amortir l'impact du **NRC**.

2. Choix et tendances:

Notre objectif premier est de concevoir un système multiprocesseur sur puce répondant à trois critères essentiels à savoir : multi-applications, haute performance et basse consommation.

L'architecture que nous proposons, se positionne dans le monde des microprocesseurs du coté des ASICs en performance et du coté des processeurs généralistes en flexibilité **(figure2)** tout en affichant une consommation permettant de l'embarquer dans des systèmes mobiles ou nomades.

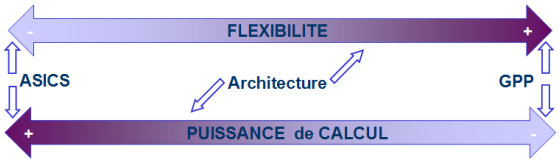

Figure 2. *Positionnement de l'architecture proposée dans le monde des microprocesseurs.*

2.1. Le défi :

Le défi que nous relevons est de concevoir une architecture performante sans pour autant être dédiée donc flexible mais sans perte en performances.

La nécessité de la flexibilité ; une flexibilité qui n'est certes pas totale, du fait que nous ciblons des domaines d'applications bien précis, introduit malgré tout des contraintes supplémentaires sur la conception de l'architecture et nous amène à prévoir des cœurs **GPP**. De même La flexibilité est réalisée par ce qu'on appelle les unités reconfigurables ; ce sont des unités dédiées et que l'on peut reconfigurer à un faible coût en vue d'une optimisation pour un traitement précis. Ces unités reconfigurables améliorent donc la performance sans perte de flexibilité.

Le caractère flot de données et calcul intensif des applications considérées, nous pousse à prévoir des blocs de calcul vectoriel afin d'optimiser des types de traitement communs à ce type d'applications telles que le produit matriciel ou le produit scalaire.

Les mots d'ordre pour avoir une flexibilité sans perte de performances et sans augmentation de la consommation ni de la complexité de l'architecture sont donc : hétérogénéité et reconfigurabilité. Des réseaux d'interconnexion flexibles permettant d'interconnecter ces composants hétérogènes prendront également place dans l'architecture proposée.

2.2. Les choix d'architecture :

Notre **MPSoCs**, est soumis à des contraintes de performances, de consommation et de flexibilité qui nous imposent des choix matériel rigoureux. A ces contraintes, liées essentiellement aux applications, viennent s'ajouter des contraintes plutôt liées à la programmation et au fonctionnement du système.

Guidés par des contraintes de programmation et d'utilisation de l'architecture nous avons fait le choix de prévoir différents types et niveaux de parallélisme ainsi que différents niveaux de contrôle.

Enfin, pour garantir une certaine performance, nous avons choisi de séparer les chemins de données et les chemins de contrôle ainsi que les espaces de stockage des données et des programmes. Des précisions sur les choix d'architecture sont présentées au paragraphe suivant.

3. Architecture:

Notre architecture sera présentée en trois niveaux correspondant à trois niveaux de parallélisme et de contrôle.

3.1. Premier niveau : MPSoC

Le premier niveau **(figure3)** que nous appellerons niveau Puce ou **MPSoC** est composé de plusieurs processeurs sur une puce, connectés via un réseau et accompagnés d'un module d'accès à une mémoire externe et un autre pour le contrôle des Entrées/Sorties permettant d'acheminer les données de et vers le **MPSoC**.

A ce niveau notre **MPSoC** est vu comme une architecture *MIMD* à mémoire distribuée où chaque processeur (**PROC i**) exécute indépendamment des autres son propre programme. Des mécanismes de synchronisation sont prévus afin de permettre une exécution coopérative entre les processeurs. Nous pouvons donc prévoir un parallélisme de programmes avec différentes granularités.

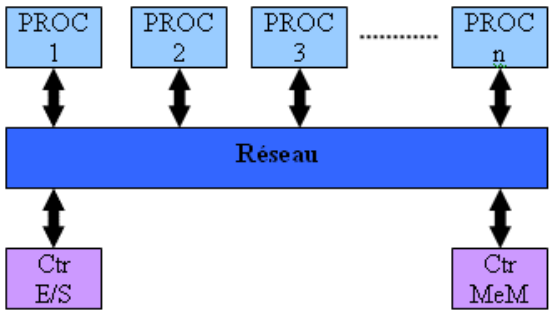

Figure 3. *Niveau MPSoC.*

Afin de palier des contraintes de sûreté de fonctionnement, le contrôle à ce niveau est totalement distribué. Pour le cas particulier du démarrage de l'application le processeur (**PROC 1**) sera choisi comme processeur maître et prendra en charge l'envoi des codes à exécuter sur chacun des autres processeurs. Dans le cas où celui-ci serait

défectueux, c'est le processeur (**Proc 2**) qui prendra en charge cette opération en inhibant le processeur défectueux et ainsi de suite. Ceci garantira une exécution qui ne sera certes pas optimale mais évitera certainement l'arrêt complet du système.

3.2. Second niveau : PROC

Le second niveau de l'architecture **(figure4)** est essentiellement composé d'un processeur de contrôle **CPU**, d'un processeur de traitement capable d'exécuter des opérations flottantes **UT**, d'une unité de traitement vectoriel **SIMD**, d'un tableau de mémoires **G-RAM** ainsi que d'autres éléments qui seront présentés dans ce qui suit.

A ce niveau, parallélisme d'instructions et parallélisme de données peuvent être entrelacés. En effet une exécution **SIMD** (parallélisme de données) peut se dérouler en même temps qu'une exécution sur l'unité de traitement **UT**.

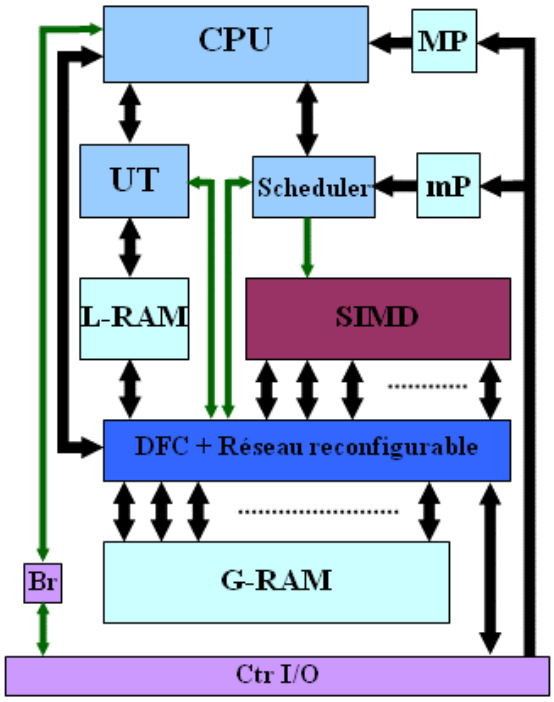

Figure 4. *Niveau PROC.*

Contrairement au niveau **MPSoC**, au niveau **PROC** le contrôle est centralisé. Le **CPU** représente le cœur du processeur, en effet ce dernier lit les instructions à partir de la **MP**[2], les décode et attribue l'exécution à l'élément de traitement adéquat à savoir l'unité de traitement **UT** ou le **SIMD**. Le **CPU** ne contrôle pas directement le module **SIMD** du processeur mais transmet une macro-instruction au **Scheduler** qui va interpréter cette macro, charger le micro-code correspondant à partir de la **mP**[3] et piloter l'exécution sur le **SIMD**. Le **CPU** s'occupe également de la préparation des données à traiter mais là aussi il ne prend pas en charge la gestion fine de cette opération mais l'attribue au **DFC**[4] qui va piloter l'accès aux différentes mémoires du système ainsi que le transfert à travers le réseau. Le **CPU** transmet juste une configuration du réseau ainsi qu'une macro-instruction de transfert mémoire.

Nous distinguons clairement une différentiation des niveaux de **contrôle et des modèles de programmation**. Etant données La complexité des architectures multiprocesseurs et les difficultés que trouvent les programmeurs à utiliser efficacement ces architectures et à tirer bénéfices de toute la puissance de calcul qu'elles offrent (cas du processeur Cell) - problème récurrent et commun à la plupart des **MPSoCs** existants-, nous proposons la différentiation des niveaux de contrôle comme un moyen de simplifier la programmation.

L'architecture est construite de manière à ce qu'une fois les macro-commandes pour le **DFC** et l'unité de traitement adéquate transmises, le **CPU** est déchargé de la suite du traitement. En effet des mécanismes de synchronisation sont prévus entre le **DFC** et les éléments de traitement. Un mécanisme de synchronisation est également prévu pour les transmissions de commandes inter-processeurs et ceci via l'élément **Br**[5] de l'architecture du **PROC**.

Nous remarquons les différents niveaux de mémorisation (l'unité **SIMD** comporte un niveau plus bas de mémorisation) ceci permet de disposer de mémoires réduites mais rapides à un premier niveau et d'un espace de stockage important mais plus lent à un deuxième niveau. Nous soulignons, enfin, l'intérêt de la séparation des mémoires de données (**L-RAM** et **G-RAM**) et des mémoires programme (**MP** et **mP**). La **G-RAM** est formée par plusieurs bancs mémoire. Nous, exposons l'intérêt de ce dispositif au paragraphe suivant.

3.3. Troisième niveau : SIMD

Le troisième et dernier niveau que nous appelons niveau **SIMD** représente une unité de traitement vectoriel. Il est formé **(figure5)** par plusieurs Processeurs Elémentaires (**PEs**) interconnectés à travers un réseau reconfigurable, comparable à celui utilisé au niveau **PROC**, avec un même nombre de mémoires à accès rapide de type « Scratch Pad Memory ».

Afin d'offrir des débits d'entrée/sortie suffisant au SIMD, nous avons prévu, au niveau **PROC**, une mémoire de donnée multi-bancs (le nombre de bancs mémoire sera égal au nombre de **PEs** du **SIMD**). Ceci permet de charger en parallèle les données sur les RAMs du **SIMD**.

[2] Mémoire de **M**acro **P**rogramme.
[3] Mémoire de **m**icro **P**rogramme.
[4] **D**ata **F**low **C**ontroller
[5] **Br**idge.

Le fonctionnement de ce système est défini de sorte qu'à un moment donné et pour un traitement donné, chaque **PE** a accès à une et une seule **RAM**.

3.4. les réseaux reconfigurables :

Comme nous l'avons souligné précédemment la flexibilité de cette architecture vient par l'aspect programmable des processeurs et également par les réseaux reconfigurables, aussi bien au niveau **SIMD** qu'au niveau **PROC**.

La reconfiguration des réseaux ne s'effectue pas au cycle près mais entre deux opérateurs. Le réseau reconfigurable au niveau **SIMD** permet de particulariser ce dernier en fonction du type d'opérateur. Le réseau reconfigurable entre le SIMD et la G-RAM, quant à lui permet d'éviter les mouvements de données.

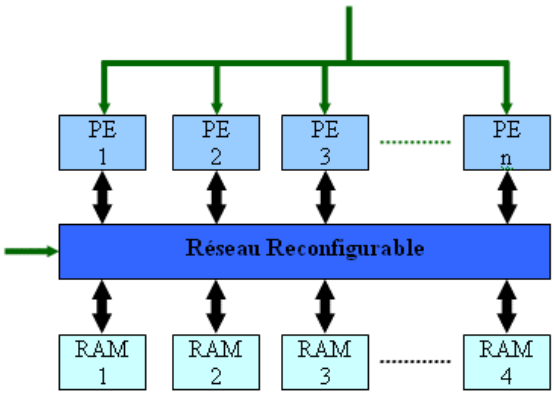

Figure 5. *Niveau SIMD*

4. Outils de programmation (SpearDE):

Développé au sein du département **ESS** « Embedded Software & Solutions » de **TRT** « THALES Research & technology », **SpearDE** est un atelier d'aide à la parallélisation et au placement de programmes avec une forte orientation flots de données. **SpearDE** permet d'avoir une vue claire sur le placement d'une application donnée sur une architecture multiprocesseur ainsi que sur les mouvements des données entre les différents éléments de l'architecture. Ainsi **SpearDE** permet, non seulement de, vérifier ou de valider l'adéquation entre un programme et une architecture en faisant varier le programme et l'architecture afin bien évidement d'atteindre des performances optimales, mais aussi de générer le code d'une application pour une cible donnée. Dans ce dernier cas des tests de performances peuvent être réalisés directement sur l'architecture ciblée ; après toute modification **SpearDE** permettra de générer à un moindre coût le code de l'application à exécuter. Dans le cas ou l'architecture cible n'existe pas, nous sommes dans une configuration d'exploration d'architecture, **SpearDE** permet de la simuler. Il intègre, en effet, un simulateur se fondant sur le moteur de simulation *SystemC*. Le simulateur **SpearDE** opérant au niveau transactionnel permet d'avoir, à moindre coût, une idée très proche des performances finales du système [Len 03].

SpearDE a été utilisé pour modéliser l'architecture proposée ainsi qu'une application de traitement radar (**STAP**). Il a également permis de placer efficacement et simplement cette application sur l'architecture et de simuler son exécution.

5. Une application à flot de données (STAP) :

Le **STAP** (Space Time Adaptive Processing) [Mel 04] est une technique qui utilise les deux dimensions espace et temps pour optimiser l'élimination des bruits et du fouillis[6] ainsi que la détection des cibles à faible vélocité et à faible altitude. De plus le **STAP** utilise des filtres adaptatifs recalculés à chaque étape de l'algorithme (en générant différentes matrices de covariance). Le **STAP** affiche ainsi une meilleure élimination (conjointe) du bruit et du fouillis ainsi qu'une robustesse et un pourcentage de détection accrus par rapport aux techniques conventionnelles [Mel 00].

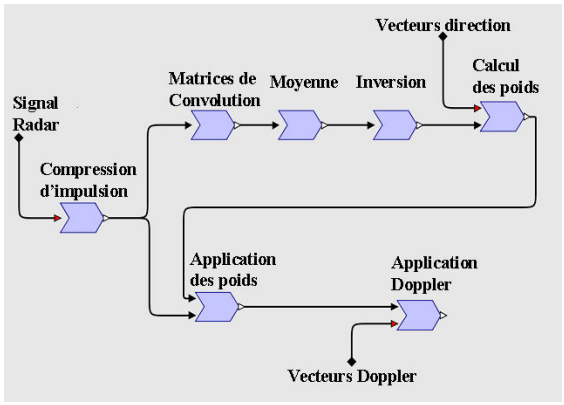

Figure 6. *Algorithme STAP.*

5.1. Description de l'application :

Le **STAP** est un algorithme qui de par sa nature (traitement espace-temps adaptatif) est excessivement coûteux autant en puissance de calcul qu'en débit d'entrée/sortie. Plusieurs recherches ont été effectuées afin de concevoir de nouveaux algorithmes inspirés du **STAP**. Ces tentatives ont abouti à des algorithmes non optimaux mais affichant des coûts beaucoup plus raisonnables. En effet, des calculs fait sur un exemple d'algorithme **STAP** montrent que nous pouvons passer d'une puissance de calcul nécessaire de l'ordre d'une dizaine de TeraFlops pour un **STAP** optimal dit *Fully Adaptive STAP* [Mel 04] à une puissance de l'ordre d'une centaine de GigaFlops pour un **STAP** « allégé » dit *partially*

[6] Echo indésirable présent dans le faisceau réfléchi d'un radar.

Adaptive STAP [Muy] [Mah 96]. C'est ce type d'algorithme qui est aujourd'hui employé dans les Radars utilisant la technique adaptative espace-temps [Gri 00].

L'algorithme **STAP** traite un tableau de données en trois dimensions : La dimension espace est obtenue par la multiplication du nombre d'antennes, la dimension temps est obtenue en envoyant plusieurs impulsions identiques pendant une rafale, et la troisième dimension correspond au nombre de cases distance (Range Gates) obtenues par l'échantillonnage du signal reçu.

Nous présentons ici l'algorithme **STAP (figure6)** retenu ainsi qu'une brève description des étapes de cet algorithme.

L'algorithme **STAP** retenu comporte sept étapes :

1-Compression d'impulsion : consiste en la convolution du signal reçu par le signal émis.

2-Matrices de Convolution : consiste en la génération des matrices d'auto-convolution, à partir des données convoluées à l'étape précédente.

3-Moyenne : consiste en un simple calcul de moyenne arithmétique sur les matrices de convolution.

4-Inversion : consiste en l'inversion des matrices de convolution. Ces matrices étant hermitiennes, la méthode de Gauss-Jacobi peut être utilisée.

5-Calcul des poids : consiste en la multiplication des vecteurs directions (un vecteur par direction) par les matrices inversées afin de générer les poids à appliquer sur les données convoluées à la première étape du traitement.

6-Application des poids : consiste à filtrer les données en entrée en leur appliquant les poids correspondant à chaque direction afin de déterminer la direction de la cible.

7-Application Doppler : appliquer des filtres doppler (un par vitesse), afin de déterminer le doppler et ainsi la vitesse de la cible.

A l'issue de ces étapes, le résultat obtenu par le STAP permet de déterminer la direction, la vitesse et la distance de la cible par rapport à l'émetteur.

5.2. Placement du STAP sur l'architecture proposée :

Nous présentons dans ce paragraphe un placement de l'application **STAP** sur l'architecture proposée. Nous commençons d'abord par préciser les différents choix qui ont été pris pour l'architecture ainsi que les paramètres retenus pour l'application.

5.2.1. ***Les choix relatifs à l'architecture*** *:*

Les contraintes liées à l'occupation de l'espace sur la puce ainsi que la consommation d'énergie ont été décisives pour le choix de l'architecture. Le tableau suivant **(tableau1)** résume ces choix.

Nombre de PROCs	8
Taille du SIMD par PROC	8
Taille de la G-RAM par PROC	512Ko
Taille d'une RAM dans un SIMD	4Ko
Taille de la L-RAM par PROC	64Ko
Taille de la MP	32Ko
Taille d'une mP	32Ko
Fréquence	800 Mhz

Tableau 1. *Les Choix relatifs à l'architecture.*

L'étude fine des éléments de calcul du **MPSoC** au niveau de chaque processeur est en cours de réalisation. Afin de présenter le placement de l'application STAP sur le **MPSoC**, nous prenons les hypothèses suivantes : un Processeur Elémentaire du **SIMD** exécute une opération entière par cycle ; l'unité de traitement du processeur (**UT**) exécute quatre opérations flottantes et deux opérations entières par cycle.

Symbole	Valeur	Description
Nsa	8	Nombre d'antennes
NrecTotal	128	Nombre de pulse émis par rafale
Ncd	111	Nombre de Cases Distance
CPI	10.24	Durée d'une rafale en **ms**
Lenght_pulse	16	Taille du pulse émis en nombre de points échantillonnés
Nrec	128	Nombre de Dopplers considérés pour la détection
Ntheta	1	Nombre de directions considérées (directions où regarde le radar)
Ntt	10	Paramètre de l'algorithme : donne le nombre d'échantillons sélectionnés. Plusieurs exemples d'algorithme STAP prennent Ntt = 3.

Tableau 2. ***Paramètres de l'application STAP.***

5.2.2. Les choix relatifs à *l'application :*

Les paramètres de l'application correspondent aux données du radar ainsi qu'à des paramètres propres à l'algorithme choisi. Ils sont regroupés dans le tableau suivant **(tableau2)**.

Nous considérons une seule direction, ce qui représente un cas réel de l'utilisation du **STAP**, à savoir déterminer avec précision la position et la vitesse radiale relatives d'une cible dont la direction est connue préalablement.

5.2.3. Placement de l'algorithme sur l'architecture :

L'algorithme **STAP** présenté au paragraphe précédent demande une puissance de calcul de 141 Gops. Sachant que le MPSoC proposé peut exécuter 76 Gops nous avons choisi de simuler le STAP sur une carte munie de 4 **MPSoCs** interconnectés à travers un anneau de type « anneau à jetons ».

Les données numérisées sont transmises à l'architecture à raison d'un tableau (*NrecTotal*× *Ncd*) par antenne. Nous avons donc 2 tableaux par puce **(figure7)**. Ces tableaux sont repartis de la même manière sur chaque puce. La suite du traitement étant identique sur toutes les puces nous nous ne décrivons ici que la répartition des tableaux 1 et 2 sur la première puce (la répartition des tableaux de 3 à 8 est facilement déductible par analogie avec celle des tableaux 1 et 2).

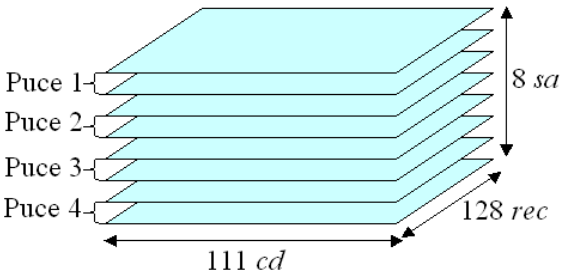

Figure 7. *Découpage des données en entrée sur l'architecture.*

Chaque tableau est réparti sur 4 processeurs. Le découpage se fait selon l'axe des récurrences (impulsions) ; chaque processeur reçoit donc un tableau de (32 *rec*[7] × 111 *cd*[8]) sur lequel est exécutée la première opération à savoir la compression d'impulsion, le signal émis étant distribué avant le début du traitement sur tous les processeurs de l'architecture. Les tableaux reçus sont stockés dans la **G-RAM** à raison de 4 tableaux par banc. L'exécution est faite en parallèle sur tous les processeurs sur le module **SIMD**. Le résultat de cette opération, un tableau de (32 *rec* × 96[9] *cd*), est stocké dans la **G-RAM**. L'opération suivante, c'est-à-dire le calcul des matrices de convolution, crée 119[10] matrices de convolution (80 × 80[11]) à partir d'une matrice (8 *sa*[12] × 128 *rec*), ceci nécessite donc que chaque processeur ait une vue des données répartie en (*sa* × *rec*). D'ou la nécessité d'une transposition globale des données (Corner Turn) contenues dans l'ensemble des processeurs de l'architecture. Chaque processeur effectuera, pour ceci, deux types de communications : des communications intra puce et des communication extra puce (à travers l'anneau). Les communications intra puce sont faites séquentiellement au sein d'une même puce et en parallèle entre les puces du système. Les communications extra puce sont faites séquentiellement entre tous les processeurs. Des requêtes de synchronisation sont émises au début et à la fin de chaque communication.

Après l'étape de communication chaque processeur dispose d'un tableau de (8 *sa* × 128 *rec* × 3 *cd*). Les tableaux sont partagés selon l'axe des récurrences sur les bancs de la **G-RAM**. Nous avons donc un tableau de (8 × 16 × 3) par banc. Les deux étapes « calcul des matrices de convolution » et « moyenne » sont fusionnées afin d'économiser l'espace mémoire. En effet le calcul des matrices de convolutions génère des données de 18.3 Mo par processeur réduites par l'opération moyenne à 153 Ko. Ces matrices sont par la suite inversées localement sur chaque processeur. Nous avons donc, au terme de cette étape et sur chaque processeur 3 matrices de convolution inversées de taille (80 × 80).

Une étape de diffusion est nécessaire avant de terminer le traitement, celle-ci sert à diffuser les vecteurs directions sur les processeurs. Une fois les vecteurs direction dans la **G-RAM** chaque processeur passe à l'étape de calcul des poids puis les applique sur les données convoluées. Une dernière étape de diffusion précède l'étape finale de l'algorithme, celle-ci concerne les vecteurs Doppler. Ceci permet à chaque processeur d'exécuter la dernière étape de l'algorithme. Cette chaîne de traitement se termine par l'envoi des résultats vers l'extérieur. Les puces sont alors prêtes à exécuter le même traitement sur le reste du flot. Ce placement est optimal car il permet d'avoir une seule étape d'échange de données entre les processeurs (lors de la transposition des données).

6. Conclusions et perspectives :

Le MPSoC présenté est destiné aussi bien au traitement de signal qu'au traitement d'image. Nous avons présenté notre proposition d'architecture ainsi qu'un exemple de placement d'une application de traitement de signal complexe. Le même travail d'étude et de placement est en cours de réalisation pour une application de traitement d'image

[7] Récurrence
[8] Case distance
[9] Correspond à *Ncd* − *Lenght_pulse* + 1
[10] Correspond à *NrecTotal* − *Ntt* + 1
[11] Correspond à *Nsa* × *Ntt*
[12] Antenne

(Segmentation d'image et extraction de contour en temps réel). De plus, des simulations **SystemC** de l'exécution des deux applications de traitement de signal et de traitement d'image sur l'architecture proposée sont en cours de réalisation.

Nous avons présenté dans cet article une proposition pour une architecture **MPSoC** adaptable à différents types d'application. Les simulations en cours et futures permettront de préciser les choix réalisés et de valider les concepts proposés. Nous chercherons également à affiner les évaluations de consommation, de puissance de calcul et de complexité de cette architecture.

REFERENCES

[ARM 05] *ARM11 MPCore Processor Technical Reference Manual*. Copyright 2005. All rights reserved ARM.

[CRA] *CT3400 Multiprocessor DSP*. CRADLE Technologies.

[Gri 00] *Space-Time Adaptive Processing in Airborne Radar Systems*. Llloyd J. Griffiths, Paul M. Techau, Jameson S. Bergin, and L. Bell. May 2000 in proceedings of IEEE 2000 International Radar Conference (RADAR 2000), Alexandria, VA, pp, 711-716.

[Gsc 06] *Synergistic Processing in Cell's Multicore Architecture*. M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Watson Research Center 2006.

[Hoe 05] *The Cell Processor*. T. Hoefler. Chaos Communication Congress (Dec 2005).

[Kah 05] *Introduction to the Cell multiprocessor*. J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. IBM J. RES. & DEV. VOL. 49 NO. 4/5 JULY/SEPTEMBER 2005.

[Len 03] *An industrial perspective: A pragmatic high end signal processing design environment at Thales*. Eric Lenormand, Gilbert Edelin. 3rd international workshop on synthesis, architectures, modeling and simulation, Samos, 2003.

[Mah 96] *Space-Time Adaptive Processing on the Mesh Synchronous Processor*. Janice S. McMahon, and Ken Teitelbaum, ipps, p. 734, 10th International Parallel Processing Symposium (IPPS '96), 1996.

[Mel 00] *Space-time adaptive radar performance in heterogeneous clutter*. William L. Melvin, IEEE Trans. AES, Vol. 36, No. 2, April 2000, pp. 621-633.

[Mel 04] *A STAP Overview*. William I. Melvin. IEEE A&E Systems Magazine Vol. 19, No. 1.

[Myu] *Parallel Implementation of a Class of Adaptive Signal Processing Applications*. Myugho Lee, Wenheng Liu, and Viktor K. Prasanna. Department of Southern California Los Angeles, CA 90089-2562.

[Pen 05] *Cell Architecture and Compilation Techniques PACT 2005 Tutorial, September 2005*. Peng Wu, M. Gschwind, K. O'Brien, and A. Eichenberger (www.ibm.com).

[Phi 02] *Nexperia PNX1300*. Philips.

[ST 04] *Nomadik - Open multimedia platform for next generation mobile devices*. ST Microelectronics.

[TI 06] *OMAP2430 applications processor*. 2006 Texas Instruments Incorporated (www.ti.com/omap).