ARCHITECTURES – SYSTÈMES EMBARQUÉS

Améliorer les performances d'un système grâce à une architecture multiprocesseur hétérogène

Cet article fait illustration d'une architecture mise en œuvre dans un système-sur-puce (SoC) pour application spatiale, où la répartition des tâches et des algorithmes associés se fait en cohérence avec les spécificités des ressources disponibles, avec un gain remarquable sur les performances et la sûreté de fonctionnement du système constitué.

DR

NICOLAS VAUCHER ET ALAIN HOUELLE (ADVANCED ELECTRONIC DESIGN - AED)

Cofondateur en 1997 de la société Advanced Electronic Design (AED), avec Alain Houelle (photo de droite), Nicolas Vaucher est docteur en microélectronique de l'université Pierre-et-Marie-Curie.
AED développe des solutions matérielles et logicielles embarquées intégrant processeurs et systèmes de traitement de signal, et propose des formations dans ses domaines d'expertise.

Classiquement, la première solution pour améliorer les performances d'un système à base de microprocesseur est d'augmenter sa fréquence de fonctionnement. Cependant, cette augmentation entraîne automatiquement une surconsommation d'énergie qu'il faudra fournir et dissiper. Cette option est bien souvent incompatible avec les systèmes embarqués. Dans le cas de traitements multitâches, une solution plus adaptée consiste à employer une architecture multicœurs afin d'améliorer les performances globales du système. Néanmoins, il n'est pas forcément judicieux, d'une part, que les processeurs soient identiques, et d'autre part, que le système soit symétrique. Dans le cadre d'une étude financée par le Cnes (Centre national d'étude spatiale), la société melunaise AED a réalisé, puis évalué les performances d'un système multiprocesseur hétérogène basé sur un processeur Leon pour l'exécution du logiciel de vol d'un satellite et sur un processeur Java (JAP) pour l'exécution d'un interpréteur d'*On Board Control Procedures* (OBCP). Les OBCP sont des procédures logicielles exécutées pendant la vie du satellite pour réaliser des tâches diverses de gestion des sous-systèmes (contrôle thermique, déploiement d'appendices…), des charges utiles (reconfiguration, gestion d'anomalies), mais qui, contrairement au logiciel de vol, doivent pouvoir être facilement chargées en vol, exécutées et remplacées. Elles requièrent donc un environnement d'exécution spécifique, historiquement, à la charge du processeur central. Le but de cette architecture était donc de décharger le processeur Leon de l'exécution des OBCP et de l'interpréteur associé – améliorant ainsi les ressources disponibles pour le logiciel de vol –, et de fournir un environnement dédié et performant pour l'exécution des OBCP.

Sans interpréteur, plus de performances et un moindre coût

Développé par l'Agence spatiale européenne (Esa), le processeur Leon possède une architecture RISC 32 bits compatible avec le Sparc V8. Le processeur JAP est capable d'exécuter le *byte code* Java éliminant ainsi l'intervention d'un interpréteur ou d'un compilateur, consommateur de ressource et impliquant coût de développement et de qualification. Une fois le *byte code* généré par le compilateur Java, les fichiers « .class » sont directement exécutés par le processeur JAP sans aucune manipulation. Le processeur JAP rassemble donc tous les mécanismes nécessaires à la gestion rapide des *bytes codes* du langage Java. Son architecture regroupe une unité en virgule fixe 32 bits basée sur une pile sécurisée et une unité arithmétique en virgule flottante 32/64 bits au format IEEE754.
Langage de programmation Objet, Java est particulièrement efficace pour réaliser des OBCP. En effet, dans ce langage, un OBCP sera représenté simplement par une classe Java regroupant variables, méthodes et *threads*. Les avantages que procure Java comme langage de programmation des OBCP sont nombreux. Mais le plus important est sûrement la sécurité de fonctionnement. Le concept d'exception du langage Java assure un comportement totalement contrôlé même en cas d'erreur. En effet, dans ce cas, l'exécution est automatiquement déroutée vers une exception sans perturber les autres OBCP. De plus, le processeur JAP apporte des mécanismes supplémentaires de sécurité afin de garantir l'intégrité des différentes piles Java de chaque *thread* OBCP.
Un langage « objet » tel que Java permet aussi de fournir naturellement des moyens de communication inter OBCP très riches et sécurisés. Une richesse basée sur le fait que les objets peuvent regrouper des variables de natures très hétérogènes (entier, nombre flottant, tableau, liste, etc.) ainsi que des fonctions. Un OBCP pourra ainsi transmettre ou récupérer des « méthodes » de traitement ou des paquets de données intrinsèquement

structurés par le langage lui-même. Sur le caractère sécurisé mis en évidence, il faut garder à l'esprit que les objets sont naturellement gérés par le langage Java et que leur transmission est atomique. Enfin, la machine virtuelle Java suit en permanence l'évolution d'un objet et peut si nécessaire le détruire si ce dernier n'est plus utilisé (fonction « ramasse-miettes » ou « *garbage collector* »).
Afin de mesurer l'impact réel de l'exécution de procédures OBCP par le JAP sur le processeur Leon, un SoC (*System on Chip*) a été développé et implanté sur une carte FPGA Xilinx. Son architecture, illustrée sur la figure en page XX, est composée des éléments principaux suivants : un processeur Leon, un processeur JAP, un contrôleur AMBA esclave APB pour la communication entre les deux processeurs et un contrôleur AMBA maître AHB pour l'accès du JAP à la mémoire SDRam.
La communication entre les deux processeurs est réalisée soit par l'envoi de messages au travers de mémoires FIFO, soit par l'utilisation d'une mémoire partagée. Enfin, le processeur Leon dispose d'un contrôle complet sur le JAP et peut exécuter les quatre commandes suivantes : reset du processeur, démarrage du processeur, arrêt du processeur et reprise de l'exécution.
L'objectif du SoC est bien sûr d'améliorer les

PERTINENCE DU CHOIX D'UNE ARCHITECTURE MULTIPROCESSEUR HÉTÉROGÈNE

Le tableau ci-dessous présente les performances du processeur JAP en fonction du paramètre du limiteur. La valeur de 100 % correspond à l'absence de limitation. 5 % indique que le processeur ne dispose que de 5 % de la bande passante pour les accès à la mémoire SDram. Les trois programmes Java testés sont :
→ **Convolution** - Calcul de convolution bidimensionnelle sur des images de 512 par 512 points sur un noyau 3x3, pour des applications de filtrage, effectuant des opérations intensives de multiplication, accumulation et déplacement de données.
→ **Correction** - Un calcul intensif de correction de trajectoire comprenant des multiplications et des additions en virgule flottante.
→ **Programme de synchronisation Java** - Il consiste à tester la communication entre *thread* en mode synchronisé. Deux *thread* s'échangent des informations en se transmettant un nouvel objet afin de solliciter au maximum le « *garbage collector* ».
Le troisième programme de test représente le pire cas en terme de consommation de bande passante qu'un programme Java puisse atteindre : création/allocation d'objet, synchronisation de deux *thread* (synchro de processus) et déclenchement du « ramasse-miettes » (*garbage collector*).
Le tableau ci-dessous montre qu'avec seulement 10 % de bande passante sur la mémoire SDram, le processeur JAP est encore capable de restituer plus de 50 % de performances (sur un programme comme Synchro par exemple) et même jusqu'à 100 % sur certaines fonctions de calcul arithmétique (correction).

PERFORMANCES DU PROCESSEUR JAP EN FONCTION DU PARAMÈTRE DU LIMITEUR

LIMITEUR (ACCÈS AUTORISÉ)	CONVOLUTION (MAC/s)	ACCÈS AUTORISÉ (%)	CORRECTION (Flop/s)	ACCÈS AUTORISÉ (%)	SYNCHRO (Inst/s)	ACCÈS AUTORISÉ (%)
100%	850000	100	800000	100	25000000	100
15%	720000	85	800000	100	21000000	84
10%	500000	59	800000	100	12000000	48
5%	265000	31	800000	100	6500000	26

MAC : Multiplication Accumulation ; *FLOP : Opération en virgule flottante ; INST : Instruction assembleur ou* byte code

performances du calculateur principal d'un satellite, mais surtout d'offrir les mécanismes nécessaires à la ségrégation des deux processeurs. En effet, le processeur JAP ne doit en aucun cas perturber le logiciel de vol exécuté par le processeur Leon. Toute tentative d'altération, par le processeur JAP, de la mémoire utilisée par le Leon a donc été rendue impossible. Qui plus est, un mécanisme de quota d'accès à la mémoire a été positionné afin de limiter la bande passante de la mémoire consommée par le processeur JAP lors de l'exécution d'un programme Java.

Une sobriété manifeste du processeur Java sur la bande passante mémoire

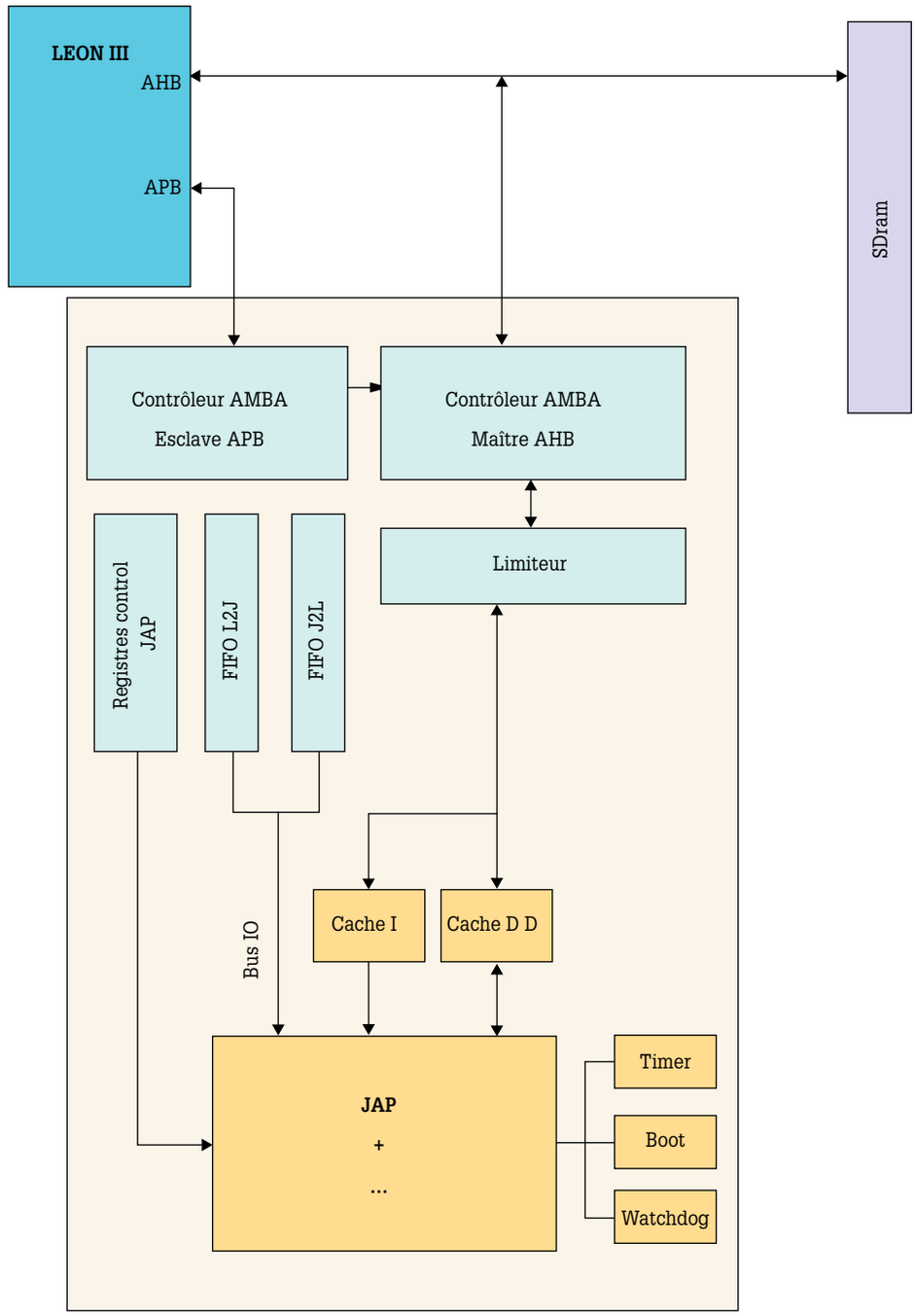
Les campagnes de tests réalisées (cf. encadré) ont démontré la pertinence de cette architecture multiprocesseur hétérogène. En effet, les tests ont mis en évidence qu'avec seulement 15 % de bande passante sur la mémoire SDram, le processeur JAP est capable de restituer plus de 85 % de ses performances et même jusqu'à 100 % sur certaines fonctions de calcul arithmétique (correction de trajectoire, calcul de matrice). Ainsi, le processeur Leon dispose de 85 % de bande passante mémoire lui permettant, grâce au cache mémoire, de fonctionner à 100 % sans réduction de ses performances. De ce fait, le processeur JAP décharge le processeur Leon de façon totalement transparente et sans limiter ses performances.

Cette faible consommation de bande passante du processeur JAP s'explique par le positionnement de mémoire cache. Cependant, deux explications supplémentaires peuvent être invoquées : la première, c'est que le processeur intègre un cache pile permettant de limiter les accès mémoire lors de l'exécution du *byte code*. En effet, les *bytes codes* utilisent toujours les données disponibles sur le haut de la pile Java. La seconde raison provient de l'architecture CISC du processeur JAP. Dans ce type d'architecture, les instructions ont une taille pouvant varier de 1 à 5 octets. En moyenne, nous avons mesuré une taille de 2,5 octets. La lecture d'une ligne cache de 8 mots de 32 bits apporte donc une vingtaine d'instructions au processeur JAP. De plus, certaines instructions sont complexes et s'exécutent par plusieurs micro-instructions contenues dans le processeur. Ainsi, le code est véritablement compacté et demande donc moins de bande passante qu'un processeur RISC classique comme le Leon.

L'implantation d'un processeur compagnon JAP n'altère en rien les performances du processeur Leon, et n'impacte pas davantage l'architecture de la mémoire (puisqu'il n'est nul besoin de banc mémoire supplémentaire). De surcroît, les performances apportées par un processeur « compagnon » Java ne sont pas négligeables : à 50 MHz, le processeur JAP est capable de fournir les performances suivantes : 850 000 multiplications par seconde, 800 000 opérations en virgule flottante au format IEEE754 par seconde et 25 000 000 instructions byte code Java par seconde.

En conclusion, l'implantation d'un processeur JAP en tant que compagnon du processeur Leon permet de libérer ce dernier des tâches de gestion et d'exécution des OBCP et d'offrir sur le JAP un berceau d'exécution sécurisé et performant. Ce SoC permet d'imaginer l'exploitation d'OBCP riches et complexes, offrant des possibilités étendues de test, de reconfiguration et d'optimisation des sous-systèmes plate-forme et charge utile tout au long de la vie du satellite. ■

Architecture du SoC développé par AED



Le processeur Leon dispose d'un contrôle complet (initialisation, démarrage/arrêt et reprise d'exécution) sur le processeur Java (JAP), avec lequel il communique par l'envoi de messages au travers de mémoires FIFO, soit par l'utilisation d'une mémoire partagée.