RTOS et gestion matérielle de la mémoire

Pierre Parrend
INSA Lyon

janvier 2005

Table des matières

1 Introduction

L'évolution récente des technologies et de la demande du marché a permis l'émergence, ces dernières années, d'un grand nombre de Systèmes Embarqués. Ceux-ci deviennent de plus en plus complexe à mesure que les technologies de circuits intégrés s'affinent jusqu'à permettre la gravure de plusieurs millions de transistors par puce, et à mesure que l'évolution des réseaux permet d'augmenter presque sans limite leur interconnexion.

Ces évolutions ont rendu indispensable l'adaptation des Systèmes d'Exploitation, dont nous présenterons les grand principes, afin de les rendre compatibles avec les caractéristiques spécifiques des Systèmes Embarqués.

Le perfectionnement des Systèmes d'Exploitation, en particulier dans le domaine du temps réel, et l'intégration matérielle toujours plus forte, ont rendu possible l'émergence de systèmes tout entiers contenus dans une seule puce, les System-on-Chip. Ceux-ci sont un champ de recherche très dynamique, dans lequel de nombreux domaines restent à explorer.

Nous nous intéresserons particulièrement, au travers de l'article 'Support matériel pour gestion de la mémoire d'un Système sur Puce Multiprocesseur Embarqué Temps-réel' ('Hardware Support for Real-Time Embedded Multiprocessor System-on-a-Chip Memory Management'), de M. Shalan et V. Mooney, du Georgia Institute of technology [SM02], aux problématiques de gestion de la mémoire, en particulier au niveau matériel.

Les auteurs présentent un mécanisme de gestion de la mémoire qui, d'après eux, cumule les avantages vers lesquels les systèmes antérieurs tendent, sans les atteindre : dynamisme, déterminisme, et rapidité d'exécution des allocations mémoire. Nous verrons comment ils arrivent à cette conclusion, et quelles sont les éventuelles limites de leurs travaux.

Nous présenterons également l'aboutissement de cet article, publié ultérieurieurement : l'intégration de ce mécanisme dans un outil de co-design qui doit rendre possible l'application de ces recherches en environement de production.

2 Contexte

La thématique des Systèmes embarqués a pris une importance croissante ces dernières années, à la fois grâce aux progrès techniques qui ont rendu possible une plus grande intégration des systèmes, et à la demande du marché qui a permis le financement de ces évolutions.

Pour comprendre l'enjeu technique de l'article [SM02], qui traite de la gestion matérielle de la mémoire dans les Systèmes sur Puce, il est important de rappeler le contexte dans lequel cet article se situe : la réunion des problématiques liées aux Systèmes Embarqués, d'une part, et aux Systèmes d'Exploitation, d'autre part. Ce qui aboutit naturellement à une nouvelle génération de Systèmes d'Exploitation spécifiques. [Sagar02] présente une introduction généraliste à ces thématiques, qui permet de prendre la mesure de l'ampleur du domaine de recherche concerné.

Ce qui rend possible la réalisation de tels systèmes est la forte progression des Systèmes sur Puce, dont [LPW97] présente les évolutions récentes, ainsi que les travaux de recherche et industriels qui y sont liés.

Pour cerner plus précisément la problématique de la gestion de la mémoire dans les Systèmes Embarqués, nous présenterons ensuite un certain nombre de travaux représentatifs de ce domaine.

2.1 Les Systèmes Embarqués

Les Systèmes Embarqués sont caractérisés par leur intégration à un système plus grand. Ils sont composés de un ou plusieurs processeurs, ainsi que de mémoire de plus ou moins grande taille. On recontre des mémoires ROM, pour les programmes à exécuter, et RAM, pour le stockage temporaire des données. [Sagar02]

La figure 1 montre le principe de fonctionnement d'un Système Embarqué. Les informations en entrée viennent de capteurs, de sondes, de boutons de contrôle, ou sont des signaux de communication. En sortie, ce sont des actionneurs, des écrans d'afichage, ou des signaux de communication.

2.2 Les Systèmes d'Exploitation

Les Systèmes d'Exploitation (SE) servent à rendre disponible les éléments matéri els d'un système informatique pour les applications. En fonction du contexte d'exécution, des objectifs du SE, les types de tâches qu'il supporte, son architecture, peuvent varier, même si son rôle reste toujours globalement identique.

Quatre types de Système d'Exploitation existent :

- mono-utilisateur, mono-tâche (Palm-OS),
- mono-utilisateur, multi-tâche (Win98, MacOS),
- multi-utilisateur (Unix, Windows-NT),
- Systèmes d'Exploitation temps réel (RTOS).

Chaque type correspond à un niveau de sureté d'exécution des tâches : simple, sans

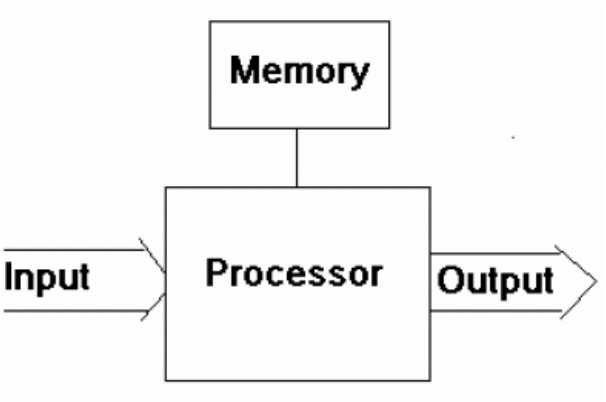

FIG. 1 – Principe des Systèmes Embarqués

perturbation des autres tâches (multi-tâches), sans perturbation des autres utilisateurs (multi-utilisateur), avec réponse en temps déterministe (RTOS).

L'architecture varie elle aussi :
- SE monolithique, peu structuré,
- SE en couche : l'accès à toutes les couches inférieures est possible (typiquement : couche matérielle, pilotes du BIOS et du Système d'Exploitation, Systèmes d'Exploitation, applications),
- SE client-serveur : le SE est organisé autour d'un micro-noyau qui exécute les tâches indispensables, ordonnancement et synchronisation des processus. L'ensemble des fonctionnalités complémentaires sont réalisées par des serveurs spécifiques, ce qui permet la scalabilité de l'OS, la distribution, la tolérance aux fautes - car une erreur dans un module n'entraine nullement d'erreur dans les modules indépendants.

Le rôle du Système d'Exploitation est multiple :
- gestion du processeur,
- gestion de la mémoire et du stockage,
- gestion des périphériques,
- interface de programmation (API),
- interface utilisateur.

Ces rôle peuvent être implémentés différemment selon l'environement du Système d'Exploitation. Un SE destiné à une machine de bureau n'aura pas les même caractéristiques que celui d'un super-calculateur, ou, à l'inverse, d'appareils légers comme les téléphones portables ou les PDA. C'est pourquoi la définition de Systèmes d'Exploitations spécifiques aux Systèmes Embarqués s'est avérée nécessaire.

2.3 Les Systèmes d'Exploitation pour Systèmes Embarqués

L'usage de Systèmes d'Exploitation est devenue nécessaire dans les Systèmes embarqués, du fait de la complexité croissante de ces systèmes (ex : Systèmes sur puce), de la présence de fortes contraintes temps réel, de la limitation des ressources disponibles, tant en mémoire qu'en énergie disponible et donc en puissance de calcul, mais également de la pression exercée par le marché sur ces produits. En effet, le temps de développement doit être raisonnable, afin de limiter le temps de mise sur le marché (time-to-market), et ainsi d'assurer le succès du produit.

Les SE à micro-noyaux sont mieux adaptés aux Systèmes Embarqués : la tolérance aux fautes doit être forte, car les conditions environnementale ne sont pas toujours optimales (ex. : systèmes pour l'automobile ou l'industrie). Ceci est permis par la possibilité de redondance, ainsi que le confinement des erreurs, qui est bien meilleurs dans un SE à micro-noyau que dans un SE plus classique. Par ailleurs, il est également possible de charger dynamiquement les modules à éxécuter, ainsi que de les distribuer, qui sont des besoins forts pour ce type de système. La figure 2 montre l'architecture d'un tel système d'exploitation.

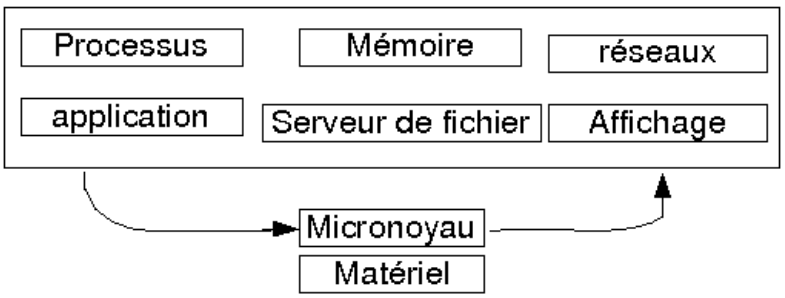

FIG. 2 – Architecture d'un SE à micro-noyau

Quelques exemples d'implémentation de Systèmes d'Exploitation pour Systèmes Embarqués :

- Inferno (Lucent) : avec sécurité intégrée,
- VxWorks, version générique,
- VxWorks, pour l'automobile,
- VxWorks, pour l'électronique gd public,
- VxWorks, pour l'industrie,
- pSOS Systems,
- Lynx Real-Time Systems (LynxOS).

Parmis les Systèmes d'Exploitations pour les Systèmes Embarqués, on distingue les SE temps-réel (RTOS - Real Time Operating Systems), qui se caractérisent par la présence de contrainte temps-réel.

2.4 Les Systèmes sur Puce

Les Systèmes sur Puce (SoC - Systems-on-Chip, ou SoS - Systems-on-Silicium) sont l'aboutissement de l'intégration croissante des Systèmes Embarqués. L'objectif est de maximiser les performances, en limitant les transferts de données et de signaux. La réduction de la taille de la gravure sur silicium permet la conception de systèmes contenus tout entier dans un puce : plusieurs processeurs, des unités fonctionnelles spécifiques, de la mémoire RAM de grande taille. [LPW97]

Des systèmes existent, qui implémentent des camescopes, de la télévision haute définition (TVHD), une radio logicielle.

2.5 Evolution des Systèmes sur Puce

La complexification des Systèmes sur Puce rend la place des RTOS de plus en plus importante. De plus, leur prix, qui reste relativement élevé, rend indispensable de bonnes performances et une bonne fiabilité, pour que ces systèmes soient compétitifs.

Ce sont des systèmes très récents, il y a donc un grand nombre de travaux en cours sur le sujet. Les directions des travaux de recherche et les évolutions apportées par les entreprises sont légèrement divergentes.

La recherche s'intéresse surtout aux aspects de co-design, et d'optimisation de l'ordonnancement.

L'industrie se concentre plus sur les performances de la mémoire, sur l'amélioration du temps de changement de contexte et du temps de latence d'interruption.

Les problèmes Un certain nombre de problèmes se posent dans cette évolution vers des systèmes plus intégrés, plus performants, et qui restent cependant suffisament rapides à concevoir.

Les principaux obstacles sont :

- RTOS et synthèse de niveau système,
- RTOS et applications embarquées,
- Débuggage (du à la complexité croissante du matériel et logiciel),
- Flexibilité, scalabilité,
- Intégration de DRAM et processeur,
- Augmentation du volume de données traitées,

C'est à l'aspect gestion de la mémoire, intégration de la RAM et du processeur, et support de grandes quantités de données, que s'intéresse l'article 'Hardware Support for Real-Time Embedded Multiprocessor System-on-a-Chip Memory Management'. Nous présenterons plus précisément les problématiques de gestion de la mémoire, avant d'étudier ce travail.

2.6 Gestion de la mémoire

La littérature traite de la question de gestion de la mémoire dans les Systèmes Embarqués sous plusieurs angles différents : langages de programmation, système, algorithmie, et implémentation matérielle. Ces travaux ne sont pas cités dans les articles proposés, mais permettent d'avoir une vue d'ensemble des problématiques liées au sujet.

Langage En matière de langages de programmation, [KW04] présente une évolution de Java temps-réel (RTSJ - Real Time Specification for Java), avec prise en compte de l'allocation mémoire à la compilation.

Système [ZS97] présente une implémentation de sémaphores prenant en compte les contraintes des Systèmes Embarqués.

Algorithmie [MRC04], entre autres, présente une optimisation de l'ordonnancement.

Matériel L'article [SM02] est un apport intéressant à la problématique de la gestion matérielle de la mémoire dans un Système Embarqué temps-réel.

3 L'Article

La partie 2 nous a permis de montrer la place qu'occupe la problématique de gestion de la mémoire dans le vaste champ des Systèmes Embarqués. L'article que nous allons présenter, [SM02], 'Support matériel pour gestion de la mémoire d'un Système sur Puce Multiprocesseur Embarqué Temps-réel' ('Hardware Support for Real-Time Embedded Multiprocessor System-on-a-Chip Memory Management'), de M. Shalan et V. Mooney, du Georgia Institute of Technology, traite plus précisément de l'implémentation d'une solution matérielle de gestion de la mémoire.

Cet article est loin d'être le premier sur le sujet, de nombreux travaux ont déjà été réalisés, qui ont tenté de mettre un place un système dynamique, rapide et déterministe. Mais aucun d'entre eux ne résoud ces trois problèmes simultannément.

La solution de [SM02] est donc originale, et complète les travaux pré-existants.

Toutefois, elle n'est pas suffisante pour proposer une méthodologie de création de Systèmes sur Puce compatible avec les besoins de l'entreprise. C'est pourquoi les auteurs ont continué leurs travaux sur le sujet, et proposé dans [SH03] une solution permettant le co-design intégrant leur proposition.

3.1 Les travaux antérieurs

Plusieurs caractéristiques déterminent les performances de l'allocation de la mémoire dans un système : sa capacité à être dynamique, et donc à s'adapter aux conditions de fonctionnement, sa vitesse, c'est à dire la latence introduite par le chargement de données en mémoire, et le déterminisme, c'est à dire la possibilité de prédire le temps nécessaire à l'exécution d'une commande d'allocation mémoire.

Le dynamisme permet à la mémoire de s'adapter aux besoins du système, par exemple de ne pas mobiliser de ressouces pour un processus en attente. Il est donc nécessaire afin de garantir un usage correct des ressources du système. Dans le cas contraire, où l'allocation mémoire serait statique, soit le système est très répétitif, du type DSP traitant un signal entrant, soit il est condamné à mobiliser d'importantes ressources qui ne sont pas utilisées, les processus (ou plus exactement les processeurs, car la mémoire est souvent allouée à un module de traitement donné) n'étant pas toujours actifs.

Ces références sont celles sur lesquelles s'appuyent notre article.

3.1.1 Allocation statique

[Wuytack99] présente un exemple d'allocation mémoire statique, avec mise en place d'une hiérarchie de mémoire. La hiérarchie de mémoire permet à un élément du système de contrôler l'accès des processeurs à la mémoire centrale du Système sur Puce, et ainsi d'améliorer ses performances en fournissant une vue générale de la configuration du système. Cette idée est l'un des piliers de la solution présentée dans [SM02].

Cette configuration est naturellement très sensible à l'évolution des contraintes et des besoins en mémoire.

3.1.2 Allocation dynamique

[Knowlton65] présente une première implantation d'une gestion dynamique de la mémoire. Son mécanisme est simple et rapide, mais rencontre des problèmes de fragmentation de la mémoire.

Pour y remédier, [Puttkamer75] propose un mécanisme qui empêche purement et simplement la fragmentation.

Ces premières propositions ne concernent pas les Systèmes Embarqués en particulier, mais l'ensemble des systèmes qui nécessitent une gestion dynamique de la mémoire. Les travaux suivants, plus récents, sont eux réalisés dans l'objectif d'améliorer les Systèmes Embarqués.

3.1.3 Accéleration de la mémoire

De nombreux travaux visent à accélerer l'accès à la mémoire.

[SLC99] présente une implémentation des fonctions d'allocation de mémoire du langage C en matériel. Malloc(), realloc(), free() sont concernées. Ceci permet d'améliorer grandement les performances des systèmes.

La même équipe, dans [Chang99], propose un accélerateur de gestion dynamique de la mémoire. Le mécanisme proposé a l'inconvénient de cacher certains blocs libres, quand le format de l'adresse du bloc ne correspond pas à une puissance de deux.

[Cam99] résoud ce problème, en permettant la détection des blocs libres d'une taille donnée.

Ces articles présentent des solutions performantes en terme de vitesse, mais on le voit, ne sont pas exempts de difficultés de mise en oeuvre, en particulier en ce qui concerne la détection de tous les blocs libres. Toutefois, s'ils proposent une allocation rapide et dynamique, il manque encore un élément indispenble aux Systèmes d'Exploitation temps-réel : le déterminisme.

3.1.4 Introduction de déterminisme

Un article un peu plus ancien, [CG96], propose une solution déterministe, qui permet d'éliminer la fragmentation, et d'avoir un temps d'allocation constant pour un bloc. Malheureusement, l'obtention de mémoire se fait bloc par bloc, ce qui a pour conséquence un temps d'allocation proportionnel au nombre de blocs de données requis.

Cette solution est donc déterministe, mais souffre de performances peu compétitives.

On le voit, le domaine de la gestion matérielle de l'allocation mémoire a été beaucoup étudié. Les mécanismes proposés apportent tous une amélioration sur un

aspect du problème, mais aucun n'arrive à concilier dynamisme, déterminisme et vitesse, car des obtacles propres aux questions de gestion de la mémoire, telle la fragmentation, rendent difficile l'implémentation d'un système qui soit optimal.

3.2 Mécanisme proposé

Les auteurs proposent dans [SM02] une solution au problème de l'allocation deterministe, rapide et dynamique de la mémoire. Il s'agir du travail réalisé par M. Shalan pendant sa thèse [Shalan03]. De plus, la solution proposée est conçue pour des puces multiprocesseurs. Le mécanisme qu'elle met en oeuvre se veut simple et puissant, de manière à ce que les performances temporelles soient bonnes.

La conciliation de toutes les propriétés du système est permise par l'utilisation d'une hiérarchie de mémoire, et l'usage d'un partitionnement particulier des blocs de données. Un élément de gestion dynamique de la mémoire, le System on-a-Chip Dynamic Memory Management Unit (SoCDMMU), est intégré au Système sur Puce.

La présence d'une hérarchie de mémoire sur deux niveaux permet d'assurer le dynamisme, et constitue une condition du détermisme.

Le partitionnement de la mémoire en blocs de données de taille constante (et adaptable) permet de compléter le mécanisme de déterminisme. Nous en expliquerons le principe ci-après.

3.2.1 Le SoCDMMU

Le SocDMMU (System on-a-Chip Dynamic Memory Management Unit) est intégré au Système sur Puce comme le montre la figure 3. Il fait l'intermédiaire entre les mémoires caches et la mémoire RAM globale.

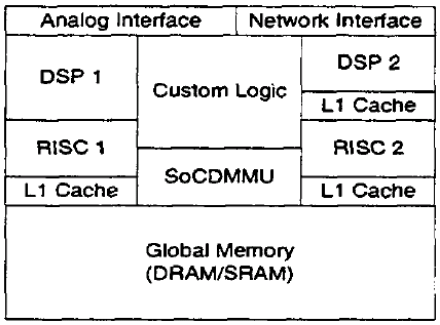

FIG. 3 – Intégration du SoCDMMU à un Système Embarqué

3.2.2 Partitionnement

Le partitionnement de la mémoire dans le Système d'Exploitation temps-réel Atalanta [Mooney02], qui est réalisé par des partitions composées de blocs de taille fixe, permet l'accès à la mémoire en temps déterministe. Pour que la taille des blocs de mémoire ne soit cependant pas rigide, les auteurs utilisent la différence entre l'espace d'adressage disponible au niveau des processeurs (4 Gbits) et la taille réelle de la mémoire globale de la puce (64 Kbits). Les adresses mémoire sont séparées en pool de 64 Kbits chacune, chacun des pools correspondant à une taille de page : le pool 0 correspond à des pages contenant 1 bloc, le pool 1 à des pages contenant 2 blocs, et ainsi de suite jusqu'à 64, comme le montre la figure 4.

Ces chiffres sont bien sûr dépendant de la configuration matérielle, et aisément généralisables.

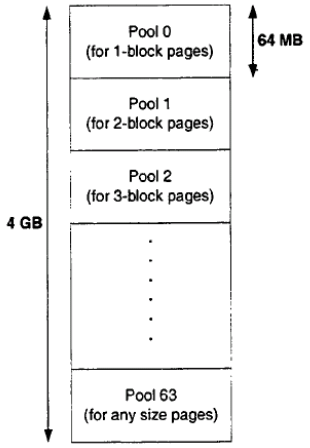

FIG. 4 – Partitionnement de la mémoire dans le RTOS Atalanta

3.2.3 Hiérarchie de mémoire

La hiérarchie de mémoire est constituée de deux niveaux : le niveau 1, dans lequel chaque processeur (les auteurs parlent de PE, Processing Element) gère l'espace mémoire qui lui a été alloué, et le niveau 2, dans lequel le SoCDMMU réalise l'allocation de la mémoire à chaque processeur.

La figure 5 montre le lien entre des deux niveaux et la mémoire globale

3.2.4 Intégration dans le RTOS

L'intégration d'un nouveau mécanisme de gestion de la mémoire peut être transparent : l'API de Atalanta pour l'allocation d'espace mémoire existe, et, si elle suffit, il n'est pas forcément nécessaire de modifier le Système d'Exploitation. La figure 6 montre les fonctions d'allocation de mémoire existantes.

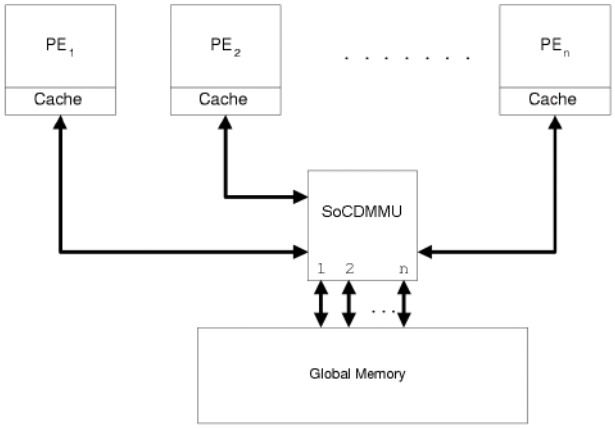

FIG. 5 – Hiérarchie de mémoire avec le SoCDMU

Toutefois, pour garantir un temps déterministe d'allocation de mémoire, le SoCDMMU doit pouvoir accéder à l'espace mémoire des processeurs. Il est donc indispensable de créer de nouvelles fonctions, qui permettent cette manipulation. La figure 7 montre ces nouvelles fonctions.

Fonction	Description
asc_partition_gain	Obtenir un bloc de mémoire d'une partition
asc_partition_seek	Obtenir un bloc de mémoire d'une partition
asc_partition_free	Libérer un bloc de mémoire
asc_partition_reference	Demande d'information sur une partition

FIG. 6 – Fonctions d'allocation mémoire existantes dans le RTOS Atalanta

Nom de la fonction	Description
asc_partition_create	Création d'une partition par le SoCDMMU
asc_partition_delete	Destruction d'une partition par le SoCDMMU
asc_memory_find	Trouver un place dans l'espace d'adressage du processeur

FIG. 7 – Nouvelles fonctions d'allocation mémoire pour le support du SoCDMMU

3.2.5 Validation

Une fois le système réalisé - ou plus exactement un prototype du système, simulé avec des outils adaptés - il est indispensable de le valider : il faut confirmer le respect du dynamisme et du déterminisme, mais il faut surtout évaluer ses performances, car c'est cet élément qui n'est pas encore formellement démontré.

Les systèmes testés sont le système présenté à la figure 8, et un système comparable ne contenant pas de SoCDMMU, mais utilisant les fonctions d'allocation optimisées de [SLC99]. En effet, comparer un système optimisé pour les Systèmes Embarqués à un système classique ne serait que peu révélateur, même s'il permettrait d'afficher de meilleurs résultats.

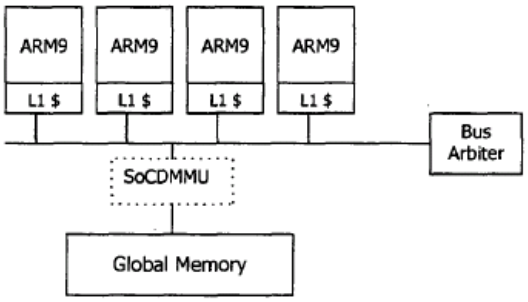

FIG. 8 – Architecture du système testé

On réalise donc un scénario dans lequel deux processus, en l'occurence un lecteur video MPEG-2 et un récepteur OFDM (type de modulation pour transmission hertzienne utilisé par exemple dans la norme 802.11a - WiFi) sont exécutés successivement. On s'intéresse au temps nécessaire pour libérer la mémoire utilisée par le lecteur vidéo et pour l'allouer au récepteur OFDM. La figure 9 montre les résultats des simulations. On observe que les gains en performance sont significatifs, tant en ce qui concerne le temps d'execution des fonctions seules (allocation, dé-allocation), qu'en ce qui concerne le temps global de traitement. Le gain total est de 440 %.

	SoCDMMU	Fonctions C	Gain
Allocation	28 Cycles	106 Cycles	3,78 x
Libération	14 Cycles	83 Cycles	5,9 x
Experience	280 Cycles	1240 Cycles	4,4 x

FIG. 9 – Résultats des tests

Ces résultats montrent que, dans le cadre de l'expérience qu'il ont réalisés, les auteurs ont atteint l'objectif annoncé : réaliser une unité de gestion dynamique de la mémoire pour des systèmes temps-réel qui allie déterminisme et performance.

3.3 Extension

L'article présenté ci-dessus montre un système efficient - qui réalise ce qu'il annonce, à savoir dynamisme et déterminisme - et efficace, c'est à dire plus rapide que les mécanismes existants. Il manque cependant une étape pour pouvoir l'utiliser dans des systèmes réels : la génération performante du code correspondant.

En effet, un système, si performant qu'il soit, n'est pas utilisable si on ne peut pas l'implémenter dans des délais qui soient compatibles avec les exigences du marché des Systèmes Embarqués.

C'est pourquoi les auteurs ont poursuivi leur projet, et réalisé un outil de co-design, qui permette la génération dynamique du système, pour un nombre quelconque de processeurs et d'éléments mémoire.[SH03]

Pour ce faire, les auteurs ont introduit un Crossbar, qui permette de relier M processeurs à N éléments mémoire. Ce Crossbar, qui permet de relier selon les besoins n'importe quel processeur à n'importe quel élément mémoire, est nécessaire si le nombre d'éléments à supporter est supérieur à la capacité d'un bus, c'est à dire en général huit.

L'architecture du système ainsi généré est présentée à la figure 10. Il s'agit d'après les auteurs des premiers travaux du genre.

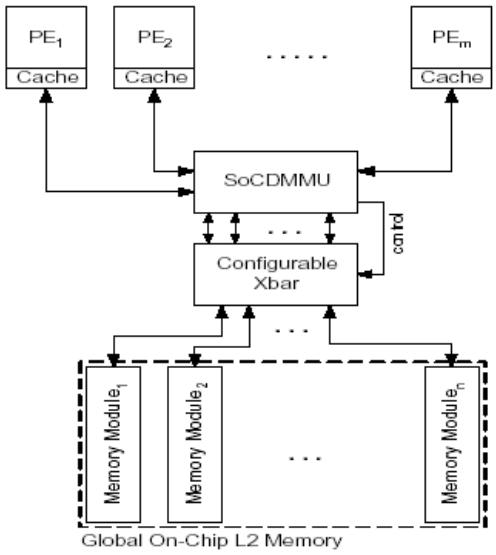

FIG. 10 – Architecture du système avec Crossbar

4 Critiques et Perspectives

4.1 Critiques positives

La solution proposée par les auteurs apporte un gain à la fois qualitatif et quantitatif par rapport aux systèmes de gestion matérielle de la mémoire. Les performances brutes sont quatre fois supérieures à celle de systèmes concurrents, et l'introduction de déterminisme et de dynamisme permet d'appliquer cette solution à des systèmes réels. D'autant plus que l'article présente en fait une partie d'un projet plus vaste, qui vise à fournir une solution de co-design de gestion de la mémoire, et donc un technologie exploitable par l'industrie.

Si les performances du système et sa faisabilité sont confirmées, il est destiné à être réutilisé, et le sera sans doute s'il bénéficie d'appuis industriels suffisants.

Ce mécanisme est donc un excellent exemple de recherche appliquée, qui s'intéresse à des aspects plus étudiés par l'industrie que par le monde de la recherche, et fournit, à partir de concepts originaux, une solution exploitable de conception de Système Embarqué.

4.2 Critiques négatives

Toutefois, quelques remarques peuvent être émises quant à ce travail.

Premièrement, le manque de transparence de l'implémentation, qui nécessite une modification de l'API du Système d'Exploitation. L'introduction de la technologie du SoCDMMU a donc des conséquences sur le SE lui-même, ce qui induit un travail préliminaire d'adaptation de celui-ci, avant de pouvoir mettre en oeuvre une gestion de la mémoire performante et adaptée aux besoins d'un Système embarqué. Mais les bénéfices potentiels en terme de fiabilité et de fonctionnalité des systèmes qui en résulteraient sont tels que ce surcoût de travail n'est pas forcément une limitation, d'autant plus que les Systèmes d'Exploitation temps-réel sont des systèmes qui sont amenés à évoluer fortement de toute façon.

Par ailleurs, une limitation au travail présenté est l'absence d'analyse critique de la solution par un pair. Il est sans doute utile de valider les résultats de l'expérience. Mais surtout, on peut imaginer qu'une modification du schéma d'allocation de la mémoire peut avoir des conséquences non prévisibles, par exemple spécifique à un type particulier de RTOS. Il peut donc être bénéfique d'étudier la portabilité de la solution proposée vers d'autres Systèmes d'Exploitations.

Ceci dit, le travail réalisé est tout à fait conséquent et intéressant, dans dans son principe que dans les perspectives qu'il offre.

4.3 Perspectives

Les auteurs présentent une nouvelle solution de gestion matérielle de la mémoire, à la fois dynamique, déterministe et rapide. Ils ont également, dans des publications ultérieures, conçu une solution de co-design qui permet de mettre en oeuvre de manière performante et efficace ce mécanisme.

Le recul manque encore pour affirmer que cette solution sera adoptée par l'industrie, et simplement qu'elle est vraiment fiable et portable sur d'autres RTOS, mais les résultats présentés sont très prometteurs.

L'aboutissement de ce travail appelle d'autres évolution dans le domaine des Systèmes sur Puce à RTOS, en ce qui concerne la mémoire la mise en place de nouveaux algorithmes d'allocation, ou plus généralement l'évolution des techniques de synthèse de niveau système.

Annexes

A Thèmes abordés

Le schéma 11 représente les différents thèmes liés à l'article présenté.
La couleur indique le degré d'importance par rapport à cet article :
– Vert, thèmes de l'article ou des articles introductifs,
– Bleu, thèmes abordés par les auteurs lors de leurs travaux, et ne concernant pas la présente synthèse sur la gestion de la mémoire et les Systèmes d'Exploitation Embarqués,
– Rouge, thèmes présenté dans cette synthèse mais non étudiés par les auteurs.

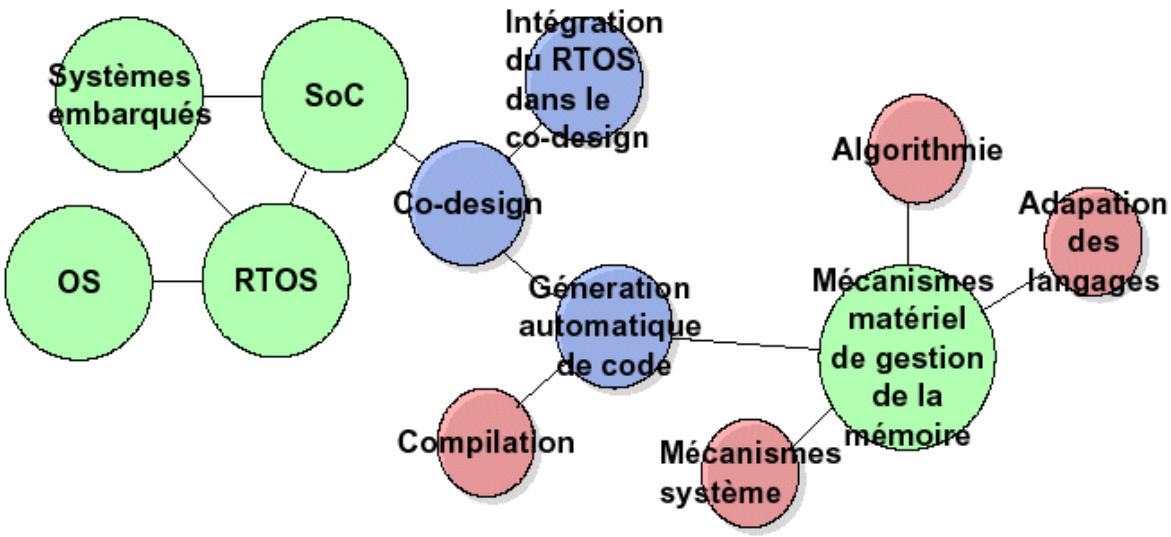

FIG. 11 – Thèmes concernant la synthèse concernant [SM02]

B Où trouver des informations ?

Si vous êtes intéressés par ce sujet, vous pourrez trouver plus d'information dans les conférences et journaux scientifiques suivants.

Les conférences :
– International Conference on Computer Design (ICCD)
– Real-Time Technology and Applications Symposium

– Euromicro Conference on Real-Time Systems (ECRTS)
– ICCD Workshop on Hardware Support for Objects and Micro architectures for Java
– Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI)

Les journaux :
– Communications ACM
– IEEE Transaction on Computers
– IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Pour retrouver ce document, ainsi que la présentation :
http ://www.rzo.free.fr/master.html

Ces références sont celles utilisées dans le cadre de cette synthèse. N'hésitez pas à vous reporter aux bibliographies des articles étudiés pour plus de précisions.

Références

[Sagar02] Embedded Operating Systems for Real-Time Applications Sagar P M (02307406) Supervisor : Prof. Vivek Agarwal. 11.2002

[LPW97] Real-Time Operating Systems for Embedded Computing Yanbing Li , Miodrag Potkonjak, and Wayne Wolf, Proceedings of the 1997 International Conference on Computer Design (ICCD '97)

[SM02] Hardware Support for Real-Time Embedded Multiprocessor System-on-a-Chip Memory Management, Mohamed Shalan, Vincent J. Mooney, CODES'02. May 6-8, 2002

[KW04] Memory Management Based on Method Invocation in RTSJ, J. Kwon, A. Wellings, Lecture Notes in Computer Science (3292) Proceedings of the OTM 2004 Workshops : Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES), 2004

[ZS97] K. M. Zuberi and K. G. Shin, "An efficient semaphore implementation scheme for small-memory embedded systems," in Proc. Real-Time Technology and Applications Symposium, pp. 25-34, June 1997.

[MRC04] Dynamic storage allocation for real-time embedded systems, 'TLSF : A New Dynamic Memory Allocator for Real-Time Systems', M. Masmano, I. Ripoll, and A. Crespo, ECRTS04

[Wuytack99] S. Wuytack et al., "Memory Management for Embedded Network Applications," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 5, pp. 533, May 1999.

[Knowlton65] K.C. Knowlton, A Fast Storage Allocator, Communications ACM, Vol. 8, October 1965, pp. 623-625.

[Puttkamer75] E. V. Puttkamer, A simple hardware buddy system memory allocator, IEEE Transaction on Computers, vol. 24, no. 10, October 1975, pp. 953-957.

[SLC99] W. Srisa-an, C. D. Lo, and J. M. Chang, "A Hardware Implementation of Realloc Function," Proceedings of WVLSI'99 IEEE Annual Workshop on VILSL April 1999, pp. 106-111.

[Chang99] J. M. Chang et al., Introduction to DMMX (Dynamic Memory Management Extension), Proceedings of ICCD Workshop on Hardware Support for Objects and Micro architectures for Java, October 1999, pp. 11-14.

[CG96] J. M. Chang and E. F. Gehringer, A High-Performance Memory Allocator for Object-Oriented Systems, IEEE Transactions on Computers, vol. 45, no. 3, March 1996, pp. 357-366.

[Cam99] H. Cam et al., A high-performance hardware-efficient memory allocation technique and design, Proceedings of International Conference on Computer Design (ICCD 99), October 1999, pp. 274-276.

[Shalan03] Dynamic Memory Management for embedded real-time Multiprocessor System on a Chip, Mohamed A. Shalan, Ph.D. Thesis, 11.2003

[Mooney02] D. Sun, D. M. Blough, and V. J. Mooney, "Atalanta : A New Multiprocessor RTOS Kernel for System-on-a-Chip Applications", Georgia Institute of Technology, Atlanta, Georgia, Technical Report GIT-CC-02-19, 2002

[SH03] M. Shalan, E. Shin and V. Mooney, "DX-Gt : Memory Management and Crossbar Switch Generator for Multiprocessor System-on-a-Chip," Proceedings of the 11th Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI'03), pp. 357-364, April 2003.