Extension Multiprocesseurs du noyau MicroC/OS-II

Yaset OLIVA, Jean-Christophe PREVOTET, Fabienne NOUVEL

Institut d'Electronique et Télécommunications de Rennes
20 Avenue des Buttes de Coësmes, 35708 Rennes
Yaset.Oliva@insa-rennes.fr, Jean-Christophe.Prevotet@insa-rennes.fr
Fabienne.Nouvel@insa-rennes.fr

Résumé – Depuis quelques années la tendance dans l'industrie du processeur a changée de cap. Aujourd'hui les améliorations en performances des systèmes électroniques, ne sont plus basées sur l'augmentation de la fréquence de fonctionnement du microprocesseur, mais sur la collaboration des multiples cœurs pour atteindre un objectif commun. Avec ce changement de technologie, les concepteurs électroniques doivent faire face à des nouveaux challenges. Dans cet article, on propose l'extension d'un système d'exploitation déjà existant, avec l'objectif de le rendre capable de gérer la tâche de porter une application sur une architecture multiprocesseur de type SMP.

Abstract – Since somes years ago, the trend in processor industry has changed course. Nowadays, the performance evolution of electronic systems is no more related to the increasing of microprocessor frequency. Instead, it depends on the results of putting several of those existing microprocessors together to achieve a common objectif. As a consequence of this change in the technology process, the designer needs to handle with new challenges. Within this paper, it is proposed the extension of an already existing Operating System. The interest is that its code has been modified, so it is capable of mapping an application onto a SMP architecture.

1 Introduction

Depuis quelques années, la fabrication d'architectures multiprocesseurs a pris une place significative dans l'industrie électronique. Aujourd'hui, les demandes du marché exigent des appareils multi-fonctions et toujours plus performants. En parallèle, force est de constater que le développement des systèmes mono-processeur atteint ses limites, lesquelles sont notamment dues à des contraintes d'intégration de plus en plus fortes et rédhibitoires en termes de coûts. L'approche actuellement suivie par les industriels consiste à concevoir de nouveaux systèmes intégrés en faisant coopérer plusieurs sous-systèmes moins rapides et moins performants. Ceci a permis la réutilisation de circuits déjà existants et conduit naturellement à une diminution du temps de conception ainsi que du coût de production.

Dans ce contexte, de nombreuses difficultés ont émergé. Afin d'exploiter la puissance de tels systèmes, il convient que tous les composants soient utilisés avec un maximum d'efficacité. Ceci implique de maitriser intelligemment le niveau de parallélisme d'exécution dans un cadre applicatif donné. Traditionnellement, les algorithmes sont décrits de manière séquentielle et l'effort nécessaire pour les paralléliser est souvent compliqué et prohibitif. D'autres problématiques peuvent également apparaitre. Parmi celles-ci, on peut noter la communication interprocesseurs et la consommation d'énergie qui deviennent un enjeu majeur.

Nos travaux de recherches se sont orientés vers des algorithmes qui permettent une répartition efficace du travail entre les processeurs d'un système possédant de multiples unités d'exécution. Ce sujet a été étudié depuis des années et de nombreuses solutions ont été élaborées. Certaines s'appuient sur des algorithmes exécutés hors ligne, au moment de la compilation [1]. D'autres travaux plus récents utilisent des algorithmes dynamiques ou des solutions quasi-statiques [2, 3] permettant d'optimiser le temps d'exécution. Dans cet article, une extension d'un système d'exploitation temps réel (RTOS [1]) pour des architectures embarquées est proposée. L'objectif principal est de disposer d'un algorithme d'ordonnancement en ligne capable d'exploiter efficacement une architecture multiprocesseur. L'article possède la structure suivante : la Section 2 explique les raisons pour les quelles nous avons fait le choix de modifier un RTOS mono processeur existant. Dans la Section 3 est spécifié le type d'architecture ciblée. En suite, les Sections 4 et 5 donnent des détails du nouveau fonctionnement ainsi que les résultats d'un exemple d'implantation respectivement. Finalement, les conclusions et perspectives sont présentées dans la Section 6.

2 Motivation

L'algorithme d'ordonnancement en ligne proposé est vu comme un service spécifique d'un système d'exploitation temps réel (RTOS). Avec l'emploi d'un tel RTOS, le partage et la gestion de ressources par des différentes applications se font possibles. La qualité d'extensible du système, est aussi grâce à la flexibilité que l'interface d'un RTOS offre à l'utilisateur. Au-

1. Real Time Operating System

trement dit, plusieurs applications peuvent être portées sur le système. Afin d'implémenter l'algorithme, il a été choisi de travailler sur un système d'exploitation existant, ouvert au sens de la disponibilité du code source. Dans la littérature, nombreux sont les RTOS qui gèrent des architectures multiprocesseurs : QNX Neutrino RTOS [4], RTEMS [5], Enea OSE [6] en sont quelques exemples. Malheureusement, la plupart de ces systèmes sont monolithiques, complexes et coûteux. Notre choix s'est porté sur un noyau très simple et très bien documenté : MicroC/OS-II [7]. Ce RTOS a une taille réduite et a été porté sur de nombreuses architectures embarquées. En revanche, ce noyau n'a pas été porté sur des systèmes possédant plusieurs unités d'exécution. Notre travail a donc consisté à modifier le code source de ce noyau afin de le rendre capable de gérer des architectures multiprocesseurs de type SMP (Symmetric Multi-Processing). Dans ce type d'architecture, tous les processeurs partagent une mémoire où se trouve le code du noyau. La raison de privilégier une approche SMP part du principe que cette approche est valable quand le nombre de processeurs n'est pas conséquent. Ceci correspond parfaitement aux architectures embarquées dont le nombre de cœurs restent encore relativement limité.

3 Type d'architecture ciblée

La configuration de l'architecture ciblée est de type SMP comme le détaille la figure 1. L'architecture est composée de N processeurs homogènes connectés par un bus global (dans un souci de simplification, seulement deux processeurs ont été représentés dans la Figure 1). Le système comprend les composants suivants : le(s) processeur(s), la mémoire partagée, des sémaphores matériels, la mémoire privée pour chaque processeur, mémoires FIFO et un horloge. Dans la mémoire partagée se trouvent les instructions du système d'exploitation ainsi que les codes de l'application fournis par l'utilisateur. L'intégrité de ces données se voit fortement menacée, lors de l'exécution en parallèle des fonctions critiques du noyau MicroC/OS-II, notamment l'ordonnanceur de tâches (Scheduler). Les sémaphores matériels sont présents pour réguler l'accès à l'information partagée. Chaque processeur est connecté aussi à un deuxième bloc de mémoire. Cette mémoire privée est utilisée pour des allocations dynamiques (Heap) et pour stocker des données spécifiques à chaque processeur, comme sont la pile d'exécution (Stack), les pilotes des possibles périphériques, le point d'entrée lors d'un démarrage et le code du programme principal. Le moyen utilisé dans le système pour la communication entre processeurs est la mémoire partagée. L'architecture contient aussi des mémoires FIFO (First In First Out), qui jouent le rôle de boîte à lettres entre certains processeurs. En séparant la mémoire pour l'exécution du code de la mémoire pour les communications entre processeurs, on réduit les possibilités d'encombrement du bus global. En outre, les FIFO garantissent un mécanisme de blocage lorsque un processeur veut lire (ou écrire) un message et la mémoire correspondante est vide (ou pleine). Ce mécanisme aurait du être implémenté si l'on utilisait la mémoire programme partagée pour les communications. L'horloge du système est connecté à un seul processeur et a la mission de lancer une interruption (Timer Tick) avec une fréquence qui dépend de l'application et qui doit être définie par l'utilisateur.

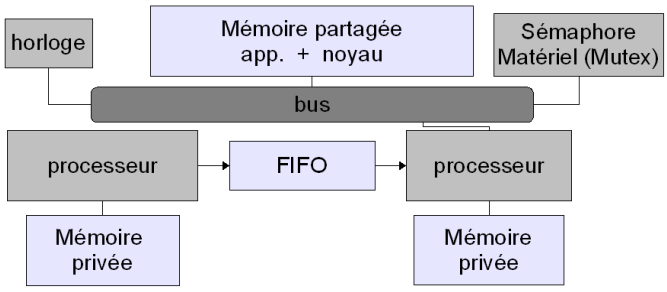

FIGURE 1 – Architecture ciblée.

4 Fonctionnement

Des nouvelles structures des données ont été ajoutées au noyau pour stocker plus d'informations sur les éléments du système. Ainsi, des objets représentant des processeurs comprennent les attributs : id, p_type et $status$. Un tableau, P_tb, stocke une instance de cette structure pour chaque processeur. L'attribut id identifie univoquement chaque processeur et sert d'indexe pour le tableau. L'attribut $status$ signale si un processeur est disponible pour exécuter une tâche ou non. D'autres informations sur les tâches ont été associées aux structures de contrôle de tâche (TCB) du noyau original : t_type, p_target, p_owner. Les attributs t_type et p_target lient la tâche à un type de processeur particulier. Ces attributs ont été ajoutés en envisageant de futures expériences avec des architectures hétérogènes. L'attribut p_owner indique quel processeur exécute la tâche actuellement.

Le système bascule entre deux modes de fonctionnement systématiquement. Le mode Maître-Esclave concerne le processus de répartition des tâches au démarrage du système et à l'exécution de la routine d'interruption de l'horloge. En mode Normal tous les processeurs accèdent à la mémoire partagée pour exécuter l'ordonnanceur de tâches et si possible par la suite, une tâche de l'application.

Dans la Figure 2, on aperçoit que le mode Maître-Esclave est activé par les signaux $Reset$ ou $Tick$. Le processeur maître réalise l'assignation des tâches vers des processeurs esclaves disponibles, en déposant dans leur mémoires FIFO un message avec l'adresse où se trouve le code d'une tâche dans la mémoire partagée. Le processus de sélection de la tâche, suit le principe de base du noyau MicroC/OS-II (i.e. la tâche la plus prioritaire). En plus, les attributs id et p_type de l'esclave exécuteur doivent être en accord avec les attributs t_type et p_target

du TCB de la tâche. Une fois que le choix est fait, les attributs p_owner du TCB et $status$ du processeur sont actualisés. Le processus se répète intégralement jusqu'à ce que toutes les tâches prêtes à l'exécution aient été traitées ou qu'il n'y ait plus d'esclaves disponibles. En outre, le maitre peut être réquisitionné si après avoir exploré toutes les possibilités d'exécution, il est le seul en mesure de répondre aux critères exigés par une tâche prête. Autrement, le maître restera en veille jusqu'à ce qu'une interruption de l'horloge du système relance le processus.

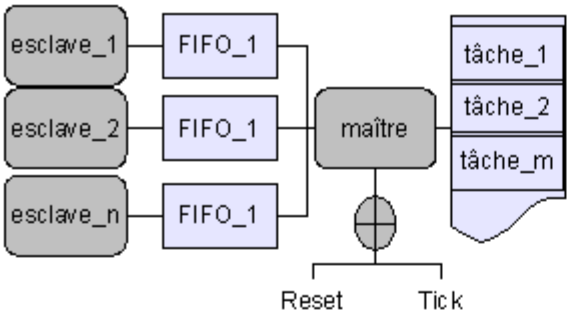

FIGURE 2 – Mode de fonctionnement Maître-Esclave.

Un esclave initie l'exécution du programme principal à partir de sa mémoire privée. Ce programme initial met l'esclave en attente jusqu'à ce qu'un message du maître soit placé dans la FIFO correspondante. A l'arrivée du message avec l'adresse de la tâche à exécuter, l'esclave vide la mémoire FIFO et dirige son compteur programme vers l'adresse indiquée. Une fois placé dans la zone publique de la mémoire partagée, l'esclave exécute le code de la tâche. La Figure 3 représente le fonctionnement en mode Normal, ceci implique des accès aux objets globaux et aux fonctions du noyau par tous les processeurs. C'est à ce moment que les sémaphores matériels jouent un rôle. Par exemple dans le cas où plusieurs processeurs aillent exécuter l'ordonnanceur, un sémaphore doit être acquit pour assurer que une même tâche ne soit pas assignée plusieurs fois. La même situation peut se présenter lorsque une tâche est suspendue en attente d'un objet de synchronisation quelconque, alors que un autre processeur est en train de libérer l'objet au même moment. Si, lors de l'exécution de l'ordonnanceur, il n'existe pas de tâches disponibles à exécuter, l'esclave abandonne la zone de mémoire partagée et retourne dans sa mémoire privée et se met en attente d'un autre message ; le maître reste en veille jusqu'à la prochaine interruption.

5 Test de performances

Dans cette section, les performances de la version étendue de MicroC/OS-II sont évaluées. Avec le seul objectif de quantifier le coût de nos modifications, un scénario purement fictif a été élaboré. Ce dernier nous a permis de comparer l'exécution de notre version du noyau avec la version originale. A l'aide de

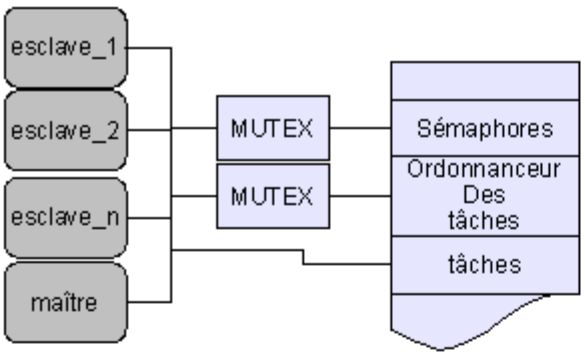

FIGURE 3 – Mode de fonctionnement Normal.

la carte de développement DE2 de chez Altera [8], nous avons implanté une architecture similaire à celle de la figure 1. Quatre processeurs NIOSII [9] ont été implantés autour de la mémoire partagée et avec ses respectives mémoires privées. En plus, un compteur de 64 bits sert à comptabiliser les temps d'exécutions désirés. Une interface JTAG-UART a été aussi ajoutée pour faire communiquer le système avec l'ordinateur hôte. Ces deux extra périphériques peuvent être accédés uniquement par le processeur qui jouera le rôle du maître dans le mode de fonctionnement Maître-Esclave.

La figure 4 détaille le graphe des tâches du scénario. Pour chaque nœud du graphe dans l'image, une tâche de MicroC/OS-II a été créée. Dans cette simulation toutes les tâches peuvent être exécutées par tous les processeurs sauf la tâche $T7$ qui doit s'exécuter exclusivement sur le processeur maître. La fonction de la tâche $T7$ se charge d'arrêter le compteur qui mesure le temps depuis le démarrage du système. Elle imprime également le résultat. L'ordre d'exécution des tâches, représenté dans le graphe par des arcs, est assuré par des objets sémaphores du noyau et des priorités. La priorité des tâches a été assignée dans l'ordre décroissant, $T1$ étant la tâche la plus prioritaire et $T7$ la moins prioritaire. De cette manière, il est certain d'arrêter le compteur à la fin de l'exécution. Tous les autres tâches exécutent un code similaire qui consiste en l'acquisition d'un sémaphore mis en disposition préalablement par la tâche prédécesseur (le cas échéant) et une boucle simulant 2 secondes d'exécution.

La table 1 compare les performances mesurées pour la version originale du noyau avec la version multiprocesseurs proposée. On y entrevoit un gain dans la version avec 4 processeurs sur la version avec 3, alors que l'ordre de précèdence représenté par les arcs, impose un paralellisme maximal de 3 tâches au même temps (dans la Figure 4 $T2$, $T3$ et $T4$). Le gain est du à la possibilité d'exécuter toutes les tâches jusqu'au moment où elles essaient d'acquérir le sémaphore. Par exemple, la version avec 3 processeurs démarre l'exécution de $T1$, $T2$ et $T3$ alors que la version avec 4 processeurs peut aussi démarrer $T4$. Même si seulement $T1$ pourra exécuter jusqu'à la fin, la prochaine exécution de $T4$ commencera à partir de l'ins-

truction suivant l'acquisition du sémaphore pour la version 4 processeurs et dès le début de la fonction pour la version 3 processeurs. La même situation se répète avec le reste des tâches.

La table montre aussi que la taille du système a augmenté d'un tiers par rapport à la taille d'origine. On considère que cette augmentation peut être acceptable si en contre-partie on obtient des améliorations jusqu'à un tiers en temps d'exécution. En fin, cette décision peut être conditionnée par le type d'application ciblée. Dans cet article, on démontre que notre ordonnanceur en ligne est efficace et permet de bénéficier des potentialités d'une architecture multiprocesseurs. Le surcout d'exécution de l'ordonnancement sur une architecture multiprocesseurs est négligeable.

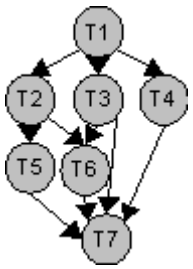

FIGURE 4 – Application fictive.

version	temps d'exécution	taille du programme
$v.o.$	14,060 s	616,51 Ko
$2CPUs$	7,595 s	809,36 Ko
$3CPUs$	6,171 s	809,85 Ko
$4CPUs$	5,145 s	810,07 Ko

TABLE 1 – Mesures de performances.

6 Conclusions et Perspectives

Avec ces travaux nous avons implanté un mécanisme d'ordonnancement en ligne qui permet de porter une application sur une architecture avec multiple processeurs. A cet effet, nous avons décidé de modifié les codes sources d'un système d'exploitation temps réel déjà existant : MicroC/OS-II. Ce choix a été conditionné par la modularité et simplicité que caractérisent ce noyau. Ayant un code ouvert et bien documenté, nous avons pu l'adapter à un type d'architecture symétrique et multiprocesseurs (SMP). Dans ce type de système, tous le processeurs accèdent à une mémoire partagée contenant les codes du noyau et de l'application. Ce fonctionnement est performant quand le nombre de processeurs est faible de manière que les accès à la mémoire ne provoquent pas d'encombrement du bus. Nous avons testé les performance de l'ordonnanceur pour une architecture réelle composée de 4 processeurs NIOSII. Les résultats ont montré que l'augmentation de la taille du code source dans la version étendue, est justifiée par une accélération de jusqu'un tiers en termes de temps d'exécution.

Nos futures travaux visent des mécanismes et applications ayant comme cible une architecture hétérogène. Nous cherchons aussi à distribuer le code du noyau entre la mémoire privé du processeur maître et la mémoire partagée, de manière à alléger encore plus les accès sur le bus global.

Références

[1] Pengcheng MU. *Rapid Prototyping Methodology for Parallel Embedded Systems. PhD thesis*. École doctorale MATISSE, 2009.

[2] Maxime PELCAT. *Rapid Prototyping and Dataflow-Based Code Generation for the 3GPP LTE eNodeB Physical Layer Mapped onto Multi-Core DSPs. PhD thesis*. École doctorale MATISSE, 2010.

[3] Luis Alejandro Cortés, Petru Eles et Zebo Peng. *Quasi-Static Scheduling for Multiprocessor Real-Time Systems with Hard and Soft Tasks*. Linköping University, Sweden 2003.

[4] *http ://www.qnx.com/products/neutrino-rtos/neutrino-rtos.html*.

[5] *http ://www.rtems.com/*.

[6] *http ://www.enea.com/Templates/Product/27035.aspx*.

[7] Jean J. LABROSSE. *MicroC/OS-II The Real-Time Kernel*. CMP Books, 2002.

[8] *http ://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html*.

[9] *http ://www.altera.com/products/ip/processors/nios2/ni2-index.html*.