EMBARQUÉ

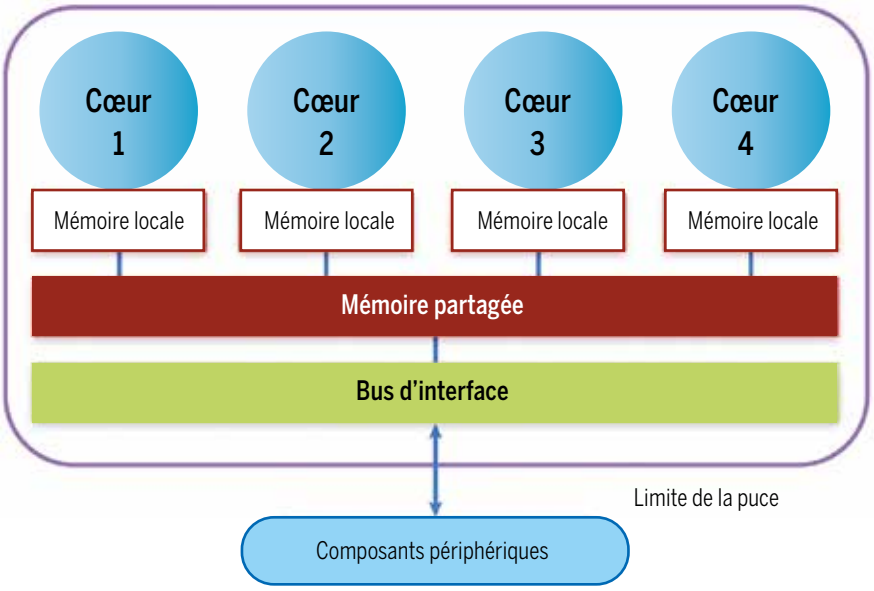
Développer des dispositifs IoT avec un OS temps-réel multicœur à ordonnancement semi-prioritaire

LA SOCIÉTÉ ESOL A DÉVELOPPÉ EMCOS, UN SYSTÈME D'EXPLOITATION TEMPS-RÉEL MULTICŒUR UTILISANT UN ALGORITHME D'ORDONNANCEMENT SEMI-PRIORITAIRE (SPBS) EXCLUSIF QUI GARANTIT LE DÉTERMINISME TEMPS-RÉEL INDISPENSABLE À DE NOMBREUX SYSTÈMES EMBARQUÉS, TOUT EN PERMETTANT D'OBTENIR DE MEILLEURES PERFORMANCES GRÂCE À L'ÉQUILIBRAGE DE CHARGES.

Masaki Gondo et Rolland Dudemaine (eSOL)
Masaki Gondo est directeur technique et vice-président d'eSOL. Il compte plus de 20 ans d'expérience dans l'architecture des systèmes d'exploitation pour l'embarqué et est l'auteur de plusieurs ouvrages et publications sur le sujet. Rolland Dudemaine est vice-président d'eSOL Europe en charge de l'ingénierie et travaille depuis près de 20 ans dans les logiciels embarqués.

La croissance exponentielle des solutions IoT (*Internet of Things*) et des applications cloud complexes ne peut se faire avec succès qu'avec l'aide de fonctionnalités nouvelles au sein de systèmes « intelligents ». Ce besoin se manifeste dans la montée en puissance de tendances comme l'edge computing (informatique en périphérie de réseau) - qui est déjà en train de migrer vers l'IA edge (intelligence artificielle en périphérie de réseau) - et la demande accrue en matière de sécurité intelligente des dispositifs IoT. Ce besoin d'informatique de pointe de l'espace IoT est partagé avec d'autres espaces d'applications nécessitant un traitement économique des données. Ce besoin est aujourd'hui satisfait par des dispositifs multicœurs.

Figure 1.- Processeur multicœur générique



Chaque cœur peut disposer d'une mémoire locale à laquelle les autres cœurs peuvent accéder et qui constitue une mémoire partagée distribuée, ou bien chaque cœur peut abriter un cache multi-niveau et une mémoire partagée globalement.

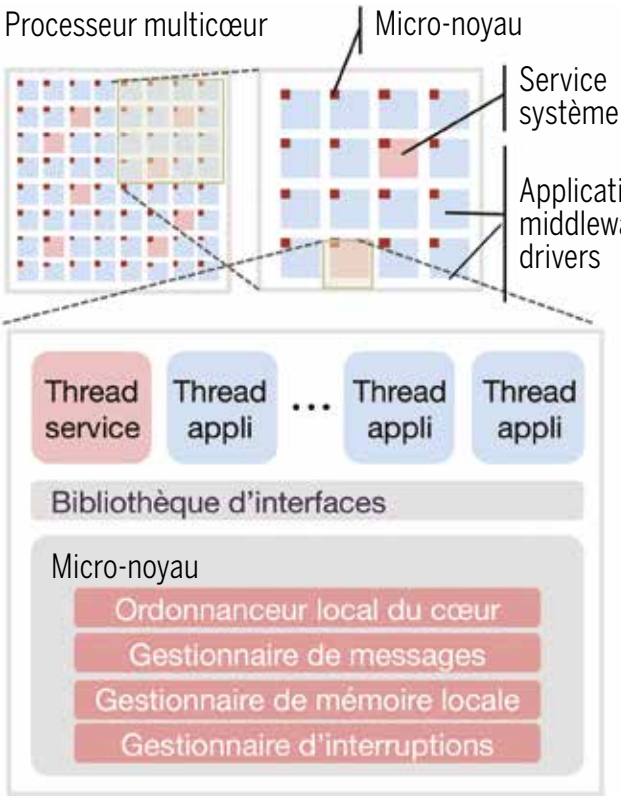
CONSIDÉRATIONS RELATIVES À L'APPROCHE MULTICŒUR

Le nombre de cœurs programmables intégrés dans une même puce est en augmentation. Si dans certains cas, il s'agit de cœurs hétérogènes, l'utilisation de cœurs homogènes se développe également. Souvent considérées comme un bon moyen d'obtenir plus de performance par watt, les solutions hétérogènes, optimales pour certaines applications, ne sont pas forcément adaptables à d'autres. Cela offre une opportunité aux solutions à cœurs homogènes qui permettent une plus grande évolutivité.

Alors qu'une puce typique est hétérogène dans son ensemble, une configuration fréquente consiste à utiliser des grappes de cœurs accélérateurs à usage général (CPU), à usage graphique (GPU) ou à usage dédié (DSP), avec plusieurs grappes de cœurs homogènes plus petits. L'ensemble est associé à un certain nombre de sous-systèmes mémoire constitués d'une mémoire locale partagée et d'une mémoire externe à laquelle les cœurs ne peuvent pas accéder directement. Chaque cœur peut disposer d'une mémoire locale à laquelle les autres cœurs peuvent accéder et qui constitue une mémoire partagée distribuée, ou bien chaque cœur peut abriter un cache multi-niveau et une mémoire partagée

globalement (*voir figure 1*).
L'augmentation du nombre de cœurs nécessite davantage de traitements logiciels et une gestion particulière des cœurs eux-mêmes. L'approche la plus courante pour gérer les logiciels consiste à utiliser un système d'exploitation, et dans ce cas, un système d'exploitation multicœur. Pour les processeurs multicœurs, dont le nombre de cœurs est le plus souvent de quatre et rarement beaucoup plus, on utilise soit un modèle SMP (multitraitement symétrique), soit un modèle AMP (multitraitement asymétrique).
Le modèle SMP n'impose pas un rôle fixe déterminé à chaque cœur du CPU, la répartition de la charge se faisant grâce au traitement d'un même programme par plusieurs cœurs. Étant donné que le SMP répartit la tâche et l'exécute dynamiquement sur plusieurs cœurs, les capacités temps-réel ne peuvent être totalement garanties et les programmes de faible priorité ne peuvent s'exécuter que lorsqu'un cœur de processeur est disponible. Par opposition, le modèle AMP est une structure logicielle de type fonction-distribution pour un système dans lequel un rôle fixe est attribué à chaque cœur, de sorte que chaque cœur traite des programmes distincts.
Les coûts de conception AMP pour l'allocation statique des tâches (ou *threads* en anglais) aux différents cœurs, la communication inter-cœurs inter-tâches, le partage des périphériques et le partage des services, peuvent être relativement lourds. Certains systèmes d'exploitation tentent de résoudre ce problème en intégrant l'approche AMP au modèle SMP, mais le nombre croissant de cœurs exige en général un modèle SMP pour les systèmes d'exploitation multicœur du point de vue du logiciel. Les problèmes liés à l'utilisation d'un système d'exploitation SMP avec un processeur multicœur sont notamment la cohérence du cache, car un grand nombre de processeurs multicœurs n'assurent pas la cohérence du cache au niveau matériel. Le modèle SMP nécessite une cohérence matérielle des caches et une mémoire partagée globalement, même si le coût du maintien de la cohérence entre plusieurs caches distants au sein d'une puce peut être élevé.
Un autre défi pour un système d'exploitation multicœur est l'ordonnancement. L'équilibrage de charges améliore le débit, mais dégrade la capacité temps-réel et le parallélisme réel, à cause de la charge et de l'incertitude supplémentaires que représente la migration des tâches. L'ordonnancement constitue un défi car il doit s'appuyer sur l'architecture d'implantation du système d'exploitation qui, justement, influe considérablement sur la conception du planificateur de l'ordonnancement.

Figure 2.- Architecture à micro-noyaux distribués



Dans un véritable système d'exploitation temps-réel multicœur, un micro-noyau réside au niveau de chaque cœur, créant ainsi un système à noyaux distribués.

ORDONNANCEMENT MULTICŒUR TEMPS-RÉEL

Une nouvelle stratégie d'ordonnancement multicœur temps-réel, dénommée SPBS (*Semi-Priority Based Scheduling*), résout les problèmes liés aux processeurs multicœurs, tels que la cohérence de cache, le partage de mémoire noyau et les transferts entre cœurs. L'application n'a pas besoin de savoir combien de cœurs sont présents, ni sur lequel de ces cœurs elle tourne. Dans un véritable système d'exploitation temps-réel multicœur, un micro-noyau réside au niveau de chaque cœur, créant ainsi un système à noyaux distribués. Ce micro-noyau comprend un gestionnaire de messages, un ordonnanceur local (limité au cœur en question) avec une gestion de tâches basique, un gestionnaire d'interruptions local (limité au cœur en question) et un gestionnaire de mémoire lui aussi local (*voir figure 2*).
Transparentes au niveau des cœurs, les API Noyau présentent une vue SMP à l'application, assurant ainsi la portabilité et l'évolutivité des performances sur un nombre de cœurs qui peut varier. Des services du système d'exploitation sont mis en œuvre sous forme de tâches serveur distribuées sur plusieurs cœurs lorsqu'un parallélisme absolu n'est pas possible. Pour les services qui nécessitent des ressources, le serveur est non seulement distribué, mais aussi hiérarchisé ; toute demande de service qui ne peut être satisfaite par un serveur local est confiée à un serveur de rang supérieur. Les tâches et les serveurs sont nommés et gérés par un *name service* (service de noms).
Le planificateur de tâches gère les tâches sur tous les noyaux d'un groupe d'ordonnancement, tandis que chaque noyau ne planifie que les tâches de son propre cœur. Un éventuel clustering, ou regroupement de plusieurs cœurs en clusters ou grappes, peut permettre de partitionner les ressources mémoires et les instances de serveur. Des primitives de communication rapide entre cœurs sont disponibles pour les tâches temps-réel dur à priorité élevée, ou pour un ensemble de tâches ayant une fréquence de communication élevée.
Réaliser un ordonnancement temps-réel avec des débits élevés implique plusieurs facteurs contradictoires. Des débits élevés nécessitent un équilibrage de charge, ce qui affecte la capacité temps-réel. Cependant, interdire l'équilibrage des charges pour assurer la capacité temps-réel entraînerait une baisse des débits moyens. Le modèle SPBS assure l'ordonnancement temps-réel des tâches de priorités supérieures, tandis que les tâches de moindre priorité sont équilibrées en fonction des tâches à effectuer (*voir figure 3*).

Examinons eMCOS, un produit développé par eSOL. eMCOS utilise un algorithme d'ordonnancement semi-prioritaire (SPBS) exclusif, qui garantit le déterminisme temps-réel indispensable à de nombreux systèmes embarqués, tout en permettant d'obtenir de meilleures performances grâce à l'équilibrage de charges. Dans le modèle SPBS, deux types de planificateurs fonctionnent en parallèle : l'ordonnanceur local à chaque cœur utilisant la priorité, et le planificateur de tâches basse-priorité, qui réalise un équilibrage de charge entre les différents cœurs disponibles en tenant compte des priorités.

L'importance de chaque tâche est exprimée sous forme de priorité de thread, une propriété dynamique considérée pour l'équilibrage de charges. En pratique, l'ordonnancement considère les threads ayant des temps d'exécution plus longs comme plus lourds, si plusieurs threads ont des priorités identiques. L'ordonnanceur fonctionne indépendamment sur chaque cœur, et utilise un algorithme classique basé sur les priorités pour sélectionner le thread à exécuter. Étant donné que chaque noyau fonctionne indépendamment, l'ordonnancement et le changement de contexte se font de manière optimisée et complètement déterministe, comme dans un système à un seul cœur, et sans avoir besoin de synchronisation entre les cœurs.

Le second planificateur distribue la charge en mesurant périodiquement l'utilisation du CPU pour chaque *thread* et il exécute les groupes de threads à faible priorité à l'aide des cœurs de processeur restants, en fonction de la priorité et du débit relatifs de ces threads. Le cas échéant, ce second planificateur peut transférer des threads à faible priorité vers d'autres cœurs moins occupés, pour plus de performance. Les threads à haute priorité sont laissés tels quels, pour conserver le déterminisme du système.

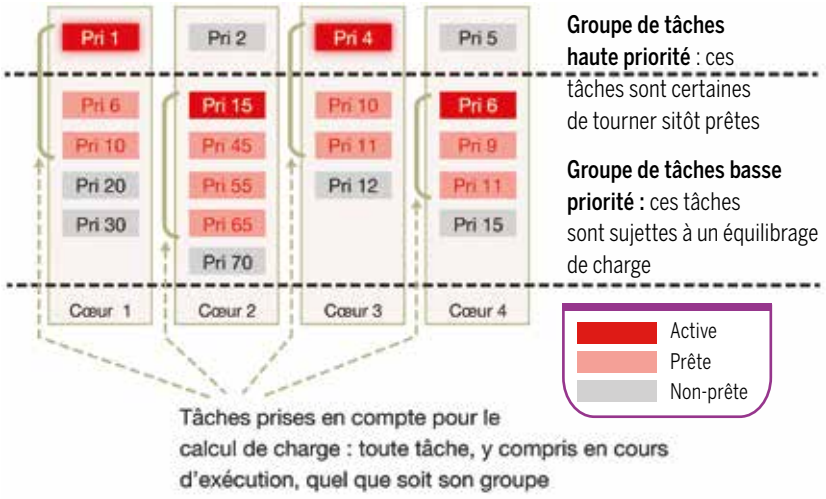
Comme il n'y a pas de migration automatique vers un autre cœur ni d'interruption d'exécution pour les tâches critiques, les développeurs peuvent calculer le temps nécessaire à chaque tâche pour achever son traitement dans les limites temps-réel requises. Une comparaison avec un prototype eMCOS montre que l'ordonnancement SPBS est nettement plus performant que les autres types d'ordonnancement, lorsque les threads présentent de fortes disparités au niveau charges de travail.

L'objectif de la stratégie SPBS est de respecter les priorités des threads, tout en maximisant le débit global. Les tâches ayant les N plus hautes priorités sont affectées à N cœurs différents, ne font l'objet d'aucune migration, et sont celles qui ont la plus haute priorité sur les cœurs auxquels ils sont affectés. Le reste des tâches fait l'objet d'un ordonnancement basé sur les priorités pour l'équilibrage de charges, dans lequel les tâches de plus haute priorité sont favorisées par rapport aux tâches de moindre priorité, tout en accordant plus de poids à la priorité des tâches qu'au nombre de tâches par cœur.

La charge de traitement que représente l'ordonnancement lui-même peut être assez importante, et aussi non-déterministe, lorsque le nombre de tâches augmente. Par conséquent, avec l'ordonnancement SPBS, le planificateur de tâches effectue des approximations, de sorte que le coût de la migration reste minimal et plus déterministe. Les éventuelles dépendances de données ne sont actuellement pas prises en compte dans l'ordonnancement des tâches. Il faut donc que les applications créent des tâches fortement dépendantes des données dans le même cluster, plutôt que de les répartir sur plusieurs grappes différentes.

L'approche de type SPBS pour un système d'exploitation temps-réel multicœur s'appuie aussi sur le nivellement du nombre de tâches, qui permet de distribuer le nombre de tâches par cœur, et sur un mapping pseudo-aléatoire, qui permet de mapper aléatoirement toute nouvelle tâche sur un cœur. L'ordonnancement 100 % préemptif basé sur des priorités fixes que l'on peut trouver normalement dans un RTOS (système d'exploitation temps-réel) commercial nécessite un échange constant de messages entre les différents cœurs, qui se traduit par une surcharge importante et nuit aux performances.

Figure 3.- Aperçu de l'ordonnancement semi-prioritaire



Le modèle SPBS (*Semi-Priority Based Scheduling*) assure l'ordonnancement temps-réel des tâches de priorités supérieures, tandis que les tâches de moindre priorité sont équilibrées en fonction des tâches à effectuer.

SOLUTION POUR L'AVENIR

Un système d'exploitation temps-réel multicœur avec une stratégie novatrice d'ordonnancement semi-prioritaire fonctionne de manière très efficace par rapport à d'autres algorithmes. La valeur optimale du facteur servant à calculer la charge de travail d'une tâche – actuellement la priorité de la tâche élevée au carré – gagnerait à pouvoir prendre en compte les propriétés d'un ensemble de tâches particulier. Cela inclut notamment la distribution des priorités de threads sur les différents nombres de tâches en tenant compte de la distribution des temps d'exécution de ces tâches.

Le volume de logiciels tournant sur des processeurs multicœurs ne cesse d'augmenter, ce qui nécessite la création et la suppression dynamiques de certaines tâches. L'optimisation de la consommation des processeurs nécessitera certainement un contrôle dynamique de la puissance fournie aux différents cœurs de manière indépendante. Cela permettra d'introduire un nouveau paramètre dans l'équilibrage de charges, indépendamment de l'optimisation du débit. L'ordonnancement semi-prioritaire peut aussi répondre à cette problématique en adaptant l'algorithme utilisé dans le second planificateur, ainsi qu'à d'autres besoins à venir. ■