Simulation d'ordonnancement temps réel avec prise en compte de l'impact des caches

Maxime Chéramy, Pierre-Emmanuel Hladik, Anne-Marie Déplanche

Simulation d'ordonnancement temps réel avec prise en compte de l'impact des caches

Maxime Chéramy*, Pierre-Emmanuel Hladik* and Anne-Marie Déplanche†
*CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
Email : maxime.cheramy@laas.fr
†LUNAM Université - Université de Nantes,
IRCCyN UMR CNRS 6597, (Institut de Recherche en Communications et Cybernétique de Nantes), ECN,
1 rue de la Noe, BP92101, F-44321 Nantes cedex 3, France

***Résumé*—Afin de prendre en charge au mieux les supports d'exécution multiprocesseurs pour les systèmes temps réel, de nombreuses politiques d'ordonnancement ont été proposées. Cependant, l'augmentation de la complexité de ces architectures rend plus difficile l'évaluation et la comparaison de ces algorithmes. Notre objectif est de faciliter ce travail en fournissant un outil de simulation qui intègre certains aspects pouvant avoir un impact important sur les systèmes réels, à savoir, les caches, la durée de prise de décision d'ordonnancement et les changements de contexte.**

I. Introduction

L'utilisation de plus en plus importante des architectures multiprocesseurs a donné un nouvel essor aux travaux liés à l'ordonnancement pour les systèmes temps réel. En effet, le multiprocesseur, en plus de l'aspect temporel, introduit la dimension spatiale au problème initial, c'est-à-dire que l'ordonnanceur doit non seulement déterminer quand exécuter les travaux, mais aussi sur quels processeurs les exécuter.

Une première approche pour aborder ce problème est de ramener un ordonnancement multiprocesseur a un ensemble de problèmes d'ordonnancement mono-processeur. C'est ce que l'on appelle l'ordonnancement par partitionnement : les tâches sont initialement affectées à des processeurs et leur ordonnancement est géré localement par une politique mono-processeur. À l'opposé, une approche appelée ordonnancement global, permet aux tâches de migrer librement d'un processeur à l'autre et donc théoriquement de mieux utiliser les ressources processeurs. Enfin, des approches intermédiaires existent qui permettent aux tâches de migrer de manière limitée afin de diminuer les surcoûts temporels liés aux migrations.

Actuellement, la majorité des travaux en ordonnancement temps réel porte sur les algorithmes et sont généralement théoriques, ignorant un certain nombre d'aspects pratiques liés au support d'exécution. Dans le cas multiprocesseur, les architectures apportent de nouvelles difficultés avec des caches partagés, de nouveaux bus de communication, des protocoles d'échange de données ad-hoc, etc. Ces architectures soulèvent aussi de nouvelles questions liées à l'implémentation : où s'exécute l'ordonnanceur ? faut-il verrouiller certaines données ? etc.

La comparaison en pratique des performances de ces nombreuses politiques d'ordonnancement dans le cadre multi-processeur est un problème difficile. Pour y répondre, une première solution est d'utiliser une architecture réelle ou un simulateur au cycle. Le résultat est certes précis, mais il est nécessaire d'implémenter les ordonnanceurs de préférence de manière optimisée, généralement dans un langage de bas niveau, et il faut disposer d'applications représentatives en grand nombre. Ce n'est donc pas adapté pour conduire de larges expérimentations ou pour comparer rapidement de nouvelles politiques. Au contraire, nous proposons d'étendre les modèles de base afin de réaliser une simulation que nous qualifierons à « grain intermédiaire ». L'avantage est qu'il est alors possible de développer des politiques dans un langage de plus haut niveau avec une API pour gérer les tâches et processeurs qui reste intuitive, de faire varier facilement les propriétés des tâches et d'obtenir des résultats rapidement.

Ainsi, notre contribution est une approche complémentaire aux analyses théoriques et expérimentales pour l'étude des politiques d'ordonnancement temps réel. L'objectif est de pouvoir analyser facilement le comportement des différentes politiques d'ordonnancement en intégrant un modèle intermédiaire du support d'exécution. Notons que nous ne cherchons pas à vérifier l'ordonnançabilité des systèmes, mais simplement à les évaluer en terme de performance.

Pour ce faire, nous avons étendu le modèle de tâches de Liu et Layland [1] pour y ajouter des informations caractérisant leur comportement mémoire. Nous proposons aussi une manière de décrire l'architecture matérielle pour préciser les diverses latences et les caractéristiques des caches. Nous nous basons ensuite sur des modèles statistiques pour prédire le taux de défaut de cache et par conséquent leur impact sur les durées d'exécution. Des durées fixes sont aussi associées à chaque prise de décision d'ordonnancement et à chaque changement de contexte. Tout ceci est intégré au sein d'un outil de simulation SimSo [2], disponible sous licence libre.

II. Motivations

Un effort de recherche important a été mené autour de l'ordonnancement temps réel multiprocesseur depuis la fin des années 1990. Des algorithmes optimaux ont été proposés (PFair et ses variantes PD et PD^2, ERFair, BF, SA, LLREF, LRE-TL, etc.) qui sont théoriquement capables d'ordonnancer

un système de tâches sans sous-utiliser les ressources de calcul [3]. Cependant, pour arriver à de tels résultats, ces algorithmes génèrent pour la majorité d'entre eux un grand nombre de prises de décisions qui mènent à beaucoup de préemptions et de migrations. Par conséquent, leur mise en œuvre s'avère problématique.

Ces observations ont permis de proposer de nouvelles politiques d'ordonnancement pour réduire les surcoûts en limitant le nombre de préemptions. D'autres travaux s'intéressent également à utiliser au mieux les caches [4], [5], [6]. Dans la communauté temps réel, la plupart des études visant à prendre en compte l'impact des caches le font à travers l'estimation du *worst-case execution time* (WCET) des tâches : calcul du *cache-related preempted delay* (CRPD) [7], injection des pénalités dans l'analyse d'ordonnancement [8], prédiction [9], etc. D'un autre côté, pour des systèmes non critiques, la modélisation des caches existe depuis longtemps et a été largement étudiée afin d'améliorer la performance des applications [10], [11]. Des études récentes se sont focalisées sur le partage de caches dans des systèmes multiprocesseurs [12], [13].

Notre principal objectif est la comparaison de ces nombreuses politiques d'ordonnancement et des variantes associées. Or, actuellement, le seul moyen d'y parvenir est de comparer les propriétés annoncées par leurs auteurs : complexité, nombre de point d'ordonnancement, taux d'utilisation, nombre de préemptions, nombre de migrations... Un tel travail de comparaison est difficile, voire impossible, puisque ces résultats ne sont valables que sous certaines conditions qui ne sont pas nécessairement partagées. De plus, la plupart des travaux ne prennent pas en compte dans leur évaluation l'impact réel des surcoûts liés à l'ordonnancement, au mieux, ils ne cherchent qu'à minimiser le nombre d'évènements sources de délai.

Une première approche pour prendre en compte les surcoûts d'un vrai système est alors d'utiliser un simulateur très précis ou une cible réelle. LITMUS^{RT} [14], développé à l'UNC, est un très bon exemple. Il s'agit d'une extension au noyau Linux qui permet d'en faire une plateforme expérimentale dédiée à l'étude d'ordonnancement temps réel. Cependant, un tel outil nécessite un investissement en temps conséquent avant de pouvoir le manipuler aisément et pouvoir développer une nouvelle politique. De plus, comme toute cible réelle, les expérimentations sont limitées à une plate-forme donnée et ne permettent pas d'étudier l'impact des paramètres du matériel sur l'ordonnancement. Enfin, en ce qui concerne les outils de simulation, citons Cheddar [15], MAST [16], Storm [17] ou encore Yartiss [18]. L'outil le plus proche du nôtre est sans doute Storm, cependant, il ne prend pas en compte les surcoûts liés à l'ordonnancement (prise de décision, changement de contexte) ni l'impact des caches.

Ainsi, notre intention est de fournir un outil de simulation prenant en compte l'impact des caches et le temps de prise de décision de l'ordonnanceur. Cet outil devrait nous permettre de comparer, dans des conditions identiques, de nombreuses politiques d'ordonnancement (partitionné, global ou hybride). L'objectif à long terme est de pouvoir dégager des tendances du type : l'ordonnanceur A se comporte mieux que l'ordonnanceur B, sauf si la taille des caches est trop petite comparée à l'empreinte mémoire des tâches. Et ainsi, de mieux cerner les points forts et points faibles des algorithmes.

III. Données d'entrée

A. Hiérarchie de caches

Avant toute chose, notons que nous ne considérons pour le moment que des caches de données utilisant la politique de remplacement *Least Recently Used* (LRU). Les caches d'instructions ne sont pas encore pris en considération, mais une partie de leur effet est cependant collectée à travers le « Base CPI » (voir ci-après).

Les caches sont organisés de manière hiérarchique ce qui est représenté sous la forme de listes. Une liste de caches est ainsi associée à chaque processeur. Les caches peuvent être partagés entre plusieurs processeurs à condition de respecter la propriété d'inclusion des caches.

Un cache est défini par une taille (en lignes), son associativité et la pénalité temporelle associée à un défaut de cache. La pénalité pour un cache Lx (pm_{Lx}) correspond au temps nécessaire pour accéder au niveau de cache supérieur voire au dernier niveau, la mémoire. Le temps nécessaire pour accéder au premier niveau de cache est pm_0.

B. Modèle de tâche

Nous utilisons le modèle de Liu et Layland [1] que nous étendons avec des informations supplémentaires afin de caractériser le comportement mémoire d'une tâche. Une tâche est alors définie par sa date d'activation, sa période et son échéance relative mais ne dispose plus d'une durée d'exécution fixe. En effet, la durée d'exécution sera calculée au cours de la simulation en fonction du contexte et à l'aide des données présentées ci-dessous.

1) Nombre d'instructions: Le nombre d'instructions exécutées par un travail. Ce nombre est fixe, mais pourrait être remplacé par un tirage aléatoire, selon une distribution donnée, pour caractériser la variabilité des exécutions.

2) Base CPI: Le nombre moyen de cycles par instruction (CPI) est un indicateur de performance. Plus le CPI est faible et plus le programme s'exécute rapidement [10]. On s'intéresse ici au nombre moyen de cycles nécessaires pour exécuter une instruction sans considérer les pénalités liées aux accès mémoire. On nommera cette donnée « base CPI » que l'on notera, pour une tâche τ : $base\_cpi_\tau$.

3) Taux d'accès mémoire: À partir d'une séquence d'instructions exécutée par une tâche τ, on définit mix_τ comme étant la proportion des instructions faisant un accès mémoire en lecture ou écriture.

4) Profil d'accès mémoire: Le nombre de lignes de cache différentes accédées entre deux accès consécutifs à une même ligne est appelé *stack distance* [11]. La figure 1 illustre cette notion.

La distribution formée par l'ensemble des *stack distances* pour une séquence d'exécution d'une tâche τ est appelée *Stack Distance Profile* (sdp_τ).

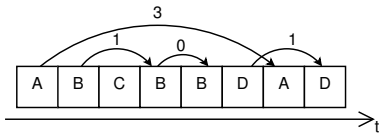

FIGURE 1. Séquence d'accès mémoire. A, B, C et D sont des lignes de cache et les nombres indiquent les distances.

C. Récolte des données d'entrée

Dans la cadre d'une expérimentation, les données qui permettent de caractériser le comportement des tâches peuvent être générées automatiquement ou obtenues depuis de vraies applications. Pour cela, le nombre d'instructions, le taux d'accès mémoire et le profil d'accès mémoire peuvent être obtenus à l'aide des logiciels MICA [19], StatStack [20] ou encore une extension à CacheGrind [21].

Le « Base CPI » nécessite l'utilisation d'un simulateur au cycle tel que gem5 [22] ou d'effectuer des mesures sur une architecture réelle.

IV. MODÈLES

Pour estimer la durée d'exécution d'un travail à partir des éléments présentés précédemment, nous disposons de multiples modèles statistiques développés par d'autres communautés scientifiques hors du cadre temps réel (évaluation de performances, optimisations à la compilation, ordonnancement, etc.).

A. Cycles Par Instruction

Le modèle de CPI que nous utilisons est celui donné en exemple dans [23] ou encore [13]. Soit, pour rappel, $base\_cpi$ le CPI sans considérer les pénalités liées aux accès mémoires et P la pénalité moyenne pour un accès mémoire. Alors, le CPI pour une tâche τ est défini par :

$$cpi_\tau = base\_cpi_\tau + mix_\tau \cdot P_\tau \tag{1}$$

P_τ est la pénalité associée à un accès mémoire et s'exprime par :

$$P_\tau = pm_0 + \sum_{x=1}^{X} mr_{\tau,Lx} \cdot pm_{Lx} \tag{2}$$

où $mr_{\tau,Lx}$ est le taux de défaut de cache pour le cache Lx pour la tâche τ, pm_{Lx} la pénalité temporelle associée à un défaut de cache sur le cache Lx, pm_0 le temps pour accéder au premier niveau de cache et enfin X est le nombre de niveaux de caches.

B. Défauts de cache de type « Capacité »

On appelle les défauts de cache de type « Capacité », les défauts de cache qui sont directement liés à la taille du cache. Autrement dit, ce sont les accès mémoires dont la dernière référence est trop ancienne pour avoir été conservée dans le cache.

Ainsi, avec un cache LRU, tous les accès mémoire se faisant à une *stack distance* supérieure à la taille du cache (C) entraîne un défaut de cache. Par conséquent, le taux de défaut de cache de type « Capacité » pour une tâche τ est :

$$mrc_\tau(C) = 1 - \sum_{i=0}^{C-1} h_\tau(i) \tag{3}$$

avec $h_\tau(i)$ la proportion des accès mémoire se faisant à une distance de i.

C. Défauts de cache de type « Chargement »

Au début, les caches sont initialement vides ce qui provoque systématiquement des défauts de cache que l'on nomme « cold misses ». De manière analogue, lorsqu'un travail a subi une préemption, une partie de ses lignes a pu être évincée par une autre tâche. On appelle cela une perte d'affinité avec le cache. Ces deux cas sont traités de la même façon.

Pendant l'exécution du système, l'état des caches (nombre de lignes par tâches et ancienneté) est estimé. Pour cela nous nous basons sur des modèles à base de chaîne de Markov inspirés de [21] que nous ne détaillerons pas ici, mais qui nous permettent d'obtenir une estimation moyenne sur le nombre de lignes chargées en fonction du nombre d'instructions exécutées (voir exemple Figure 2).

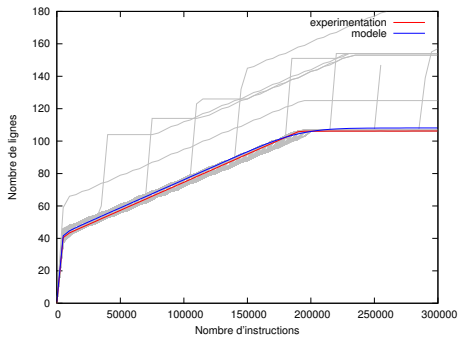

FIGURE 2. Chargement du cache en fonction du nombre d'instructions exécutées pour le bench CRC32 de MiBench. En gris les traces des différentes exécutions à partir de différents instants, en rouge la moyenne et en bleu le résultat du calcul par notre modèle.

D. Partage de cache

Lorsque plusieurs tâches s'exécutent en parallèle sur différents processeurs partageant des caches, ceux-ci ne sont généralement pas partagés de manière équitable. En effet, une tâche qui fait beaucoup d'accès mémoire aura tendance à s'accaparer une grande partie du cache en évitant à ses lignes d'en être exclues.

De nombreux modèles existent pour traiter ce cas. Dans un premier temps, nous avons retenu le modèle FOA [24] qui a l'avantage d'être rapide à calculer tout en restant relativement précis. Cependant, d'autres modèles pourraient être mis en place [13], [21], [24].

E. Avancement de l'exécution d'un travail

Les modèles ci-dessus permettent d'obtenir, en fonction d'un nombre d'instructions, le nombre de défauts de cache.

On cherche à obtenir, à partir d'une durée d'exécution, l'avancement en terme de nombre d'instructions exécutées. Or, la durée d'exécution est égale au CPI multiplié par le nombre d'instructions. Ainsi, en reprenant l'équation (1), le nombre d'instructions exécutées pendant d cycles par une tâche τ est :

$$\frac{d}{base\_cpi_\tau + mix_\tau \cdot P_\tau} \tag{4}$$

Puisque la valeur de P_τ dépend également du nombre d'instructions, on obtient un problème de point fixe qui se résout en pratique rapidement par itérations.

V. Travaux en cours et futurs

Les modèles présentés ci-dessus sont intégrés dans un simulateur baptisé SimSo [2]. Il est continuellement en évolution, mais a atteint un niveau de fonctionnalité qui le rend dès à présent utilisable pour expérimenter des politiques d'ordonnancement. Il est possible d'utiliser un mode de simulation qui se base sur le WCET ainsi qu'un mode prenant en compte l'impact des caches. Cependant, certains modèles, ainsi que leur utilisation conjointe au sein du simulateur restent à valider.

Une fois cette étape importante de validation faite, une large campagne de tests sera menée afin de comparer de nombreuses politiques.

Enfin, grâce à une meilleure compréhension des différents phénomènes observés, on espère pouvoir contribuer à améliorer certaines politiques ou à en proposer de nouvelles.

VI. Conclusion

À travers ce papier, nous avons présenté nos objectifs, à savoir la comparaison des politiques d'ordonnancement multiprocesseur en tenant compte des surcoûts liés à l'ordonnancement. L'impact des caches représente une part importante des surcoûts observés lors d'une préemption ou migration et un mauvais partage des caches peut entrainer de forts ralentissements. C'est pour cette raison que nos efforts se concentrent sur leur prise en compte au sein de la simulation.

Bien que notre démarche ne permette pas d'améliorer directement l'ordonnançabilité des systèmes, elle devrait nous permettre de mieux identifier les sources de surcoûts et par conséquent d'ouvrir la voie à leur réduction. Les bénéfices sont nombreux : réduction de l'activité des processeurs et par conséquent de la consommation et de la chaleur, meilleure réactivité du système, plus grande marge de sécurité par rapport aux contraintes temporelles.

Pour le moment, nous travaillons à la validation des modèles utilisés en comparant les résultats attendus à ceux obtenus par simulation fine. Dès que cette étape de validation sera faite, nous passerons à une étape de test et de caractérisation des politiques d'ordonnancement multiprocesseur publiées.

Références

[1] C. L. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, 1 1973.

[2] M. Chéramy, P.-E. Hladik, and A.-M. Déplanche, "Simulation of real-time multiprocessor scheduling with overheads," in *Proc. of SIMULTECH 2013 (accepted)*, 2013.

[3] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, vol. 43, no. 4, 2011.

[4] A. Fedorova, M. Seltzer, and M. Smith, "Cache-fair thread scheduling for multicore processors," Division of Engineering and Applied Sciences, Harvard University, Tech. Rep. TR-17-06, 2006.

[5] J. Anderson, J. Calandrino, and U. Devi, "Real-time scheduling on multicore platforms," in *Proc. of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2006.

[6] B. Berna and I. Puaut, "Pdpa : period driven task and cache partitioning algorithm for multi-core systems," in *Proc. of the 20th International Conference on Real-Time and Network Systems (RTNS)*, 2012.

[7] S. Altmeyer and C. Maiza Burguière, "Cache-related preemption delay via useful cache blocks : Survey and redefinition," *Journal of Systems Architecture*, vol. 57, no. 7, 2011.

[8] J. Busquets-Mataix, J. Serrano, R. Ors, P. Gil, and A. Wellings, "Adding instruction cache effect to schedulability analysis of preemptive real-time systems," in *Proc. of the Real-Time Technology and Applications Symposium 1996, (RTAS)*, 1996.

[9] D. Hardy, T. Piquet, and I. Puaut, "Using bypass to tighten wcet estimates for multi-core processors with shared instruction caches," in *Proc. of the 30th IEEE Real-Time Systems Symposium (RTSS)*, 2009.

[10] J. C. Mogul and A. Borg, "The effect of context switches on cache performance," *SIGOPS Oper. Systems Review*, vol. 25, 1991.

[11] R. Mattson, J. Gecsei, D. Slutz, and I. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal*, vol. 9, no. 2, 1970.

[12] E. Berg, H. Zeffer, and E. Hagersten, "A statistical multiprocessor cache model," in *Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software*, 2006.

[13] D. Eklov, D. Black-Schaffer, and E. Hagersten, "Fast modeling of shared caches in multicore systems," in *Proc. of the 6th International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC '11)*, 2011.

[14] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, "Litmus^{RT} : A testbed for empirically comparing real-time multiprocessor schedulers," in *Proc. of the 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, 2006.

[15] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar : a flexible real time scheduling framework," *Ada Lett.*, vol. XXIV, no. 4, 2004.

[16] M. G. Harbour, J. J. G. García, J. C. P. Gutiérrez, and J. M. D. Moyano, "Mast : Modeling and analysis suite for real time applications," in *Proc. of the 13th Euromicro Conference on Real-Time Systems (ECRTS)*, 2001.

[17] R. Urunuela, A.-M. Déplanche, and Y. Trinquet, "Storm a simulation tool for real-time multiprocessor scheduling evaluation," in *Proc. of the Emerging Technologies and Factory Automation (ETFA)*, 2010.

[18] Y. Chandarli, F. Fauberteau, D. Masson, S. Midonnet, and M. Qamhieh, "Yartiss : A tool to visualize, test, compare and evaluate real-time scheduling algorithms," in *3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2012)*, 2012.

[19] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *Micro, IEEE*, vol. 27, no. 3, 2007.

[20] D. Eklov and E. Hagersten, "StatStack : efficient modeling of LRU caches," in *Proc. of the IEEE International Symposium on Performance Analysis of Systems Software (ISPASS)*, 2010.

[21] V. Babka, P. Libič, T. Martinec, and P. Tůma, "On the accuracy of cache sharing models," in *Proc. of the third joint WOSP/SIPEW international conference on Performance Engineering (ICPE)*, 2012.

[22] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Computer Architecture News*, 2011.

[23] J. L. Hennessy and D. A. Patterson, *Computer architecture : a quantitative approach*. Morgan Kaufmann, 2011.

[24] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *Proc. of the 11th International Symposium on High-Performance Computer Architecture (HPCA)*, 2005.