Analyse de performance de méthodes de verrouillage statique de caches dans les systèmes temps-réel strict multi-tâches

Isabelle Puaut, Alexis Arnaud, David Decotigny
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cédex, France
Tel: +33 2 99 84 73 19, Fax: +33 2 99 84 71 71
e-mail: puaut@irisa.fr

Résumé

Les mémoires caches sont sources de problèmes de déterminisme dans les systèmes temps-réel strict, du fait de leur comportement dynamique et adaptatif. Ainsi, leur utilisation dans ces systèmes doit faire l'objet d'une attention particulière. Bien que de gros progrès aient été réalisés dans les dix dernières années pour analyser statiquement les pires temps d'exécution (WCET, pour Worst-Case Execution Times) de tâches pour des processeurs dotés de mémoires caches, de telles méthodes d'analyse ne sont pas toujours applicables, ou bien peuvent être excessivement pessimistes. Une autre option pour utiliser des mémoires caches dans les systèmes temps-réel strict est de verrouiller (geler) leur contenu, de telle sorte que les temps d'accès mémoire et les délais de préemption causés par les caches soient déterministes. Cet article présente et évalue les performances d'une méthode de *verrouillage statique* de caches, rendant par construction les latences d'accès à la mémoire complètement prévisibles.

Plan

Mots-clé

Systèmes temps-réel strict, architecture des processeurs, mémoires cache, verrouillage de caches, pires temps d'exécution, évaluation de performances

1 Introduction

Nous nous plaçons dans cet article dans le cadre de systèmes temps-réel strict, pour lesquels il est impératif de valider le respect des contraintes temporelles auxquelles le système est soumis, comme par exemple les échéances. De nombreuses méthodes d'analyse d'ordonnançabilité ont été conçues à cet effet [CDKM02] ; elles reposent sur une connaissance de la pire charge du système, et en particulier du pire temps d'exécution de chacune de ses tâches (WCET, pour Worst-Case Execution Time) [CPRS03].

1.1 Mémoires cache et systèmes temps-réel strict

Les mémoires caches sont des mémoires de petite taille mémorisant les informations les plus référencées par un programme, donc susceptibles d'être référencées à nouveau dans un futur proche. Bien que les mémoires caches améliorent notablement les temps d'exécution moyens des programmes, leur pire comportement, qu'il est important de connaître pour valider les systèmes temps-réel strict, est difficile à anticiper. Cette difficulté provient de la présence de conflits pour l'accès aux lignes de caches, difficiles à identifier statiquement : les conflits *intra-tâche* surviennent quand une tâche écrase ses propres informations dans le cache ; les conflits *inter-tâche* surviennent dans les systèmes multi-tâche et provoquent un temps de rechargement du cache suite à une préemption.

Une utilisation sans restriction des mémoires caches dans les systèmes temps-réel strict permet de bénéficier pleinement de l'amélioration des performances qu'elles apportent. Toutefois, analyser l'ordonnançabilité du système requiert alors une attention particulière :

- l'estimation du WCET de chaque tâche doit prendre en compte la présence de caches, et pour cela doit déterminer pour chaque accès mémoire, son pire comportement vis-à-vis du cache (succès/échec). Cette prévision doit être à la fois sûre (ne jamais prédire comme un succès un accès susceptible de provoquer un échec) et précise (par exemple, prédire tout accès comme un échec serait excessivement pessimiste). Les méthodes décrites dans [Mue00, AFMW96] réalisent une telle analyse du comportement des caches.
- dans les systèmes multi-tâches, les pires temps de rechargement du cache suite à une préemption doivent être prédits de manière à la fois sûre et précise [LHS+98].

Un autre moyen de résoudre les problèmes de déterminisme causés par les mémoires caches est d'utiliser un *partitionnement* du cache entre les différentes tâches [Kir89, SS93], réalisé grâce à un support matériel ou logiciel. Ce type de méthode élimine les conflits inter-tâches, mais pas les conflits intra-tâche. De plus, la capacité disponible par tâche se trouve réduite en comparaison avec un cache non partitionné, aboutissant ainsi à une baisse des performances globales du système.

Enfin, les méthodes de *verrouillage* de caches (*cache locking*) permettent de geler le contenu du cache (par bloc de cache ou par zone de taille plus importante) pendant une certaine durée, interdisant l'évolution du contenu de la zone verrouillée pendant cette durée [MCIBM01, PD02, VLX03]. Cette capacité de verrouillage de caches est offerte par de nombreux processeurs commerciaux (Motorola ColdFire MCF5249, Motorola PowerPC 440, Motorola MPC7451, ARM 940 et ARM 946E-S). Le contenu du cache peut

être verrouillé pendant toute l'exécution du système (verrouillage statique) [PD02] ou pendant des périodes plus courtes (verrouillage dynamique) [VLX03]. De plus, un cache verrouillé peut soit contenir des blocs pour toutes les tâches du système (verrouillage global), soit contenir des informations concernant une seule tâche (verrouillage local). Les méthodes de verrouillage global éliminent les temps de rechargement du cache après un changement de tâche, au détriment d'une capacité en mémoire cache moindre pour chaque tâche. Les méthodes de verrouillage statique des caches, sur lesquelles nous nous concentrons dans la suite de l'article, présentent un certain nombre d'attraits :

- les temps d'exécution des programmes sont améliorés en comparaison d'un programme n'utilisant pas de mémoire cache
- par construction, les temps d'accès aux informations sont *déterministes*. Il est certes toujours nécessaire d'utiliser des méthodes statiques pour prédire les WCET, mais elles sont alors plus simples, car elles n'ont plus alors à prédire le pire comportement des programmes vis-à-vis des caches, ce comportement étant connu statiquement. Le verrouillage statique peut donc être utilisé dans les circonstances ou les analyses de caches ne peuvent pas être appliquées ou sont trop pessimistes [HLTW03].
- dans les méthodes de verrouillage global, le temps de rechargement du cache suite à une préemption est nul, au détriment d'une capacité en mémoire cache moindre pour chaque tâche. Dans les méthodes de verrouillage local, la capacité disponible par tâche est plus importante, mais le délai de rechargement du cache suite à une préemption est non nul, tout en étant déterministe.
- certains systèmes permettent de verrouiller uniquement des *portions* du cache (par exemple une voie entière). Cette caractéristique peut être utilisée pour rendre déterministe les accès aux caches pour les applications temps-réel strict, que l'on associe à la portion verrouillée du cache, tout en faisant bénéficier pleinement les applications temps-réel souple des améliorations de performance.
- enfin, les méthodes de verrouillage statique de caches sont faciles à mettre en œuvre une fois le contenu du cache sélectionné; en particulier, il n'est pas nécessaire de modifier le compilateur ou le code des tâches.

1.2 Contributions et organisation de l'article

Le verrouillage de caches est couramment utilisé dans l'industrie pour garanrir un temps d'accès déterminisme à certaines portions de code critiques et de courte durée (par exemple le code des routines de traitement des interruptions). Toutefois, la sélection du contenu à verrouiller est souvent réalisée de manière *ad hoc*. Dans [PD02], nous avons proposé une méthode automatisable de sélection du contenu de caches d'instructions pour des systèmes composés de tâches périodiques. Nous nous proposons dans cet article d'en évaluer les performances en comparaison avec l'utilisation de caches non verrouillés (dynamiques). Les critères utilisés pour l'évaluation de performances sont :

- *L'ordonnançabilité du système (performances pire-cas).* L'évaluation de l'ordonnançabilité repose sur une estimation des WCET des tâches, obtenue par analyse statique de programmes. Cette évaluation nous permet d'exhiber des situations dans lesquelles les méthodes d'analyse de caches présentes dans les outils d'évaluation

statique des WCET actuels sont excessivement pessimistes. Nous montrons que dans ces mêmes situations, les évaluations de WCET sont plus précises en utilisant des caches verrouillés.

– *Les performances mesurées du système (performances cas-moyen).* Nous comparons ici les performances mesurées d'un jeu de tâches (temps d'exécution de chaque tâche, charge mesurée) avec et sans verrouillage de cache. Nous montrons ainsi que des caches verrouillés sont compétitifs avec des caches dynamiques (tout en étant légèrement moins efficaces) quand le volume d'informations manipulées par les programmes n'est pas exagérément élevé.

Les résultats expérimentaux présentés dans l'article également en exergue quelques facteurs ayant une influence sur les performances des systèmes utilisant le verrouillage statique de caches :

– *Taille des informations manipulées*. Nous montrons qu'en dessous d'un certain seuil $\frac{TailleInformations}{TailleCache}$ les performances du verrouillage statique sont compétitives avec celles de caches non verrouillés, alors qu'au dessus de ce seuil on observe d'importantes dégradations des performances.
– *Localité spatiale et temporelle des accès mémoire des tâches*. Nous montrons que meilleure est la localité des références, meilleures sont les performances globales du système.
– *Taille du cache*. Nous montrons que pour les gros caches, les temps de rechargement des caches suite à une préemption ont un impact important sur les performances, suggérant ainsi l'utilisation d'une méthode de verrouillage global. Pour les petits caches, ces temps ont une influence minime et alors le verrouillage local permet d'obtenir de meilleures performances.

Nous nous concentrons dans cet article sur le verrouillage statique de caches d'instructions. Nous avons mené une étude similaire pour les caches de données, que nous ne présentons pas ici pour des raisons de place. On se reportera à [PAD03] pour plus de détails.

L'article est organisé comme suit. Le paragraphe 2 présente l'algorithme de sélection du contenu du cache. Les conditions expérimentales sont décrites dans le paragraphe 3. le paragraphe 4 est consacré à l'évaluation de performances des méthodes de verrouillage statique de caches d'instructions. Nous concluons dans le paragraphe 5.

2 Algorithme de sélection de contenu de cache

2.1 Modèle de cache

Nous considérons des processeurs dotés de caches associatifs par ensemble de V voies. Le cache est composé de B blocs de S_B octets chacun, pour une taille totale du cache $S_C = B * S_B$ octets. Les blocs sont groupés en S ensembles de V blocs, et une information à l'adresse *ad* est stockée dans un des V blocs de l'ensemble $\lfloor \frac{ad}{S_B} \rfloor$ *modulo* S. Cette formalisation de la structure des caches inclut à la fois les caches à correspondance directe ($V = 1, S = B$), les caches associatifs par ensembles ($1 < V < B$) et les caches totalement associatifs ($V = B, S = 1$). Nous supposons par ailleurs l'existence

d'un mécanisme de *verrouillage* du contenu du cache, permettant d'inhiber le remplacement de son contenu tant que le cache reste dans l'état verrouillé. Aucune information concernant la politique de remplacement du cache n'est nécessaire au fonctionnement de l'algorithme de sélection du contenu du cache.

2.2 Modèle de tâches

Nous considérons un ensemble de N tâches périodiques $\tau_{i,\,1\leq i\leq N}$ de périodes T_i et de pires temps d'exécution C_i. Le code d'une tâche τ_i est décomposé en *lignes de programmes* $L_{i,j}$ de taille S_B; une ligne de programme est une séquence d'instructions en correspondance avec un bloc du cache d'instructions. L'algorithme de sélection de contenu du cache se base sur une connaissance pour chaque exécution de la tâche τ_i de la séquence σ_i des lignes de programmes référencées par τ_i. Nous nommons par la suite $n_{load_{i,j}}$ le nombre total de références à la ligne de programme $L_{i,j}$ dans σ_i.

2.3 L'algorithme de sélection de contenu

L'algorithme de sélection du contenu du cache, introduit initialement dans [PD02], génère un contenu de cache *global* à toutes les tâches. C'est un algorithme glouton (il ne revient pas en arrière dans son processus de décision quand il a décidé de verrouiller une ligne de programme particulière). La sélection du contenu du cache est basée sur l'idée très simple de verrouiller dans le cache les lignes de programme les plus souvent référencées, de manière à limiter la charge processeur $\sum_{i=1}^{N} \frac{C_i}{T_i}$.

Comme plusieurs tâches se partagent le cache, une *importance relative* doit être attribuée à chaque tâche pour la sélection de ses lignes de programme. Dans le modèle de tâche considéré, nous définissons pour chaque tâche τ_i son *facteur d'importance*, noté I_i, comme l'inverse de sa période T_i. Différentes définitions de l'importance d'une tâche pourraient être également utilisées pour généraliser l'algorithme à un modèle de tâches différent.

> L'algorithme procède comme suit. Soit L_s l'ensemble des lignes de programmes (pour l'ensemble des tâches) en correspondance avec les V blocs de l'ensemble s. L'algorithme choisit de verrouiller dans le cache les V lignes de programme de facteur $n_{load_{i,j}} * I_i$ les plus élevés.

L'algorithme peut être aisément transposé à des systèmes mono-tâche en éliminant le facteur d'importance I_i, obtenant ainsi une méthode de verrouillage local.

3 Conditions expérimentales

3.1 Environnement de mesure

Matériel. L'évaluation de performances a été réalisée sur un processeur MIPS R2000. De manière à pouvoir aisément modifier les paramètres architecturaux (tailles de caches,

latence d'accès, etc.), nous travaillons sur un simulateur de processeur plutôt que sur architecture réelle. Plus précisément, nous avons utilisé l'émulateur MIPS du système d'exploitation à vocation pédagogique nommé Nachos [1] de l'université de Berkeley. Nous lui avons ajouté un cache d'instructions ayant des blocs de $S_B = 16$ octets (4 instructions MIPS), et dont la structure, définie par les paramètres V et S, est paramétrable. La politique de remplacement du cache est une politique LRU (Least Recently Used). Comme seul l'impact du cache sur les performances nous intéresse ici, nous avons utilisé un modèle temporel très simple du processeur : un instruction nécessite t_{hit} cycles pour s'exécuter en cas de succès dans le cache d'instructions, et t_{miss} cycles sinon (par défaut, nous prenons $t_{hit} = 1$ et $t_{miss} = 20$).

Système d'exploitation. Les mesures de performance ont été réalisés en utilisant le système d'exploitation Nachos introduit ci-dessus, augmenté avec un ordonnancement à priorité fixe. Les priorités des tâches sont fixées selon la politique Rate Monotonic. Un système de détection de la stabilisation de la charge processeur a été mis en place de manière à éviter d'exécuter les tâches pendant une durée trop importante.

3.2 Analyse d'ordonnançabilité

Estimation des WCETs. Les WCETs des tâches sont estimés en utilisant l'outil d'analyse statique Heptane [CP01]. Heptane [2] estime les WCET récursivement en se basant sur la structure syntaxique des programmes sources C analysés (par exemple, le WCET d'une structure conditionnelle est calculé à partir des WCET de ses composants, à savoir le test de la conditionnelle et les branches *then* et *else*).

Heptane inclut des modules d'analyse statique du comportement des caches d'instructions, pipelines et prédicteurs de branchement. Comme seul le comportement des programmes vis-à-vis du cache nous intéresse ici, seul ce composant d'analyse du matériel a été conservé dans Heptane pour l'évaluation de performance.

La technique utilisée dans Heptane pour prédire le comportement des caches est une amélioration de la méthode de *simulation statique de caches* initialement proposée par F. Mueller [Mue00]. Cette technique calcule des *états abstraits de cache* ; un état abstrait représente tous les contenus possibles du cache au point considéré, et ce en prenant en compte tous les chemins d'exécution possibles. Les états abstraits de cache sont obtenus par analyse flot de données opérant sur le graphe de flot de contrôle des programmes. À partir des états abstraits, chaque instruction est classée selon dans catégorie selon les conflits pour le bloc de cache auquel elle est associée : *hit* en l'absence de conflit, *first-hit* ou *first-miss* en cas de conflits à l'intérieur de boucles, et *miss* quand aucune catégorie meilleure ne peut être trouvée [CP01]). Dans le cas de caches associatifs par ensembles, différentes catégories peuvent être obtenues pour une même instruction pouvant, d'après le mode de calcul des états abstraits de cache, se trouver dans les différentes voies de l'ensemble. Dans ce cas, pour assurer la sûreté de la méthode, l'instruction est classée dans la catégorie la plus pessimiste de toutes.

1. http://www.cs.washington.edu/homes/tom/nachos/

2. Heptane est un outil libre d'obtention des WCET par analyse statique, disponible à l'adresse *http://www.irisa.fr/aces/software/software.html*.

Pour prendre en compte la présence de caches verrouillés, Heptane a été modifié de manière à remplacer son module d'analyse de caches par un module classant les instructions selon le contenu du cache verrouillé : *hit* quand l'instruction est verrouillée dans le cache, *miss* dans le cas contraire.

Conditions d'ordonnançabilité. Les unités de mesure utilisées pour l'analyse d'ordonnançabilité sont d'une part la charge processeur pire-cas, définie par l'équation 1 ci-dessous, et d'autre part le verdict d'ordonnançabilité fourni par une analyse de temps de réponse. Dans l'équation 1, γ_i représente une borne supérieure du délai de préemption qu'une tâche τ_i impose sur les tâches interrompues par τ_i.

$$U = \sum_{i=1}^{N} \frac{C_i + \gamma_i}{T_i} \tag{1}$$

L'analyse de temps de réponse utilisée pour analyser l'ordonnançabilité du système est une extension directe des travaux de Tindell dans [TBW94] au cas de délais de préemption γ_i non nuls. Pour chaque tâche τ_i, l'analyse opère en considérant les interférences produites par les tâches strictement plus prioritaires que τ_i. Le temps de réponse de τ_i, noté R_i, est alors le point fixe de la série donnée dans l'équation 2 ci-dessous, dans laquelle $hp(\tau_i)$ dénote l'ensemble des tâches de priorités strictement supérieures à celles de τ_i (dans l'équation, les tâches sont supposées avoir des priorités distinctes).

$$\begin{aligned} w_i^0 &= C_i \\ w_i^{n+1} &= C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil * (C_j + \gamma_j) \rightarrow R_i \end{aligned} \tag{2}$$

L'ordonnançabilité du système est alors vérifiée en comparant R_i avec l'échéance de la tâche τ_i lorsque la série converge.

Lorsque un verrouillage global est utilisé, γ_i est nul. Sinon, γ_i prend en compte le temps de rechargement de tous les blocs de cache après une préemption.

3.3 Procédure d'expérimentation

Pour un ensemble de tâches périodiques donné, nous procédons en deux étapes :

1. La première étape sélectionne un contenu pour le cache verrouillé (obtention des séquences d'adresse mémoire puis application de l'algorithme de sélection de contenu). Les séquences d'adresse mémoire σ_i utilisées pour la sélection de contenu peuvent sont obtenues en exécutant le programme sous Nachos.
2. La deuxième étape est l'évaluation de performances proprement dite et est décomposée en deux parties :
   - L'évaluation de l'ordonnançabilité du système avec et sans verrouillage de cache. Les unités de mesure utilisées sont alors le verdict d'ordonnançabilité du système (équation 2) et la pire charge processeur calculée grâce à l'équation 1.

– L'évaluation des performances effectives (mesurées) du système. Les unités de mesure utilisées sont alors les temps d'exécution mesurés des tâches, et la charge CPU résultante, tous deux obtenus par instrumentation sous Nachos.

Les expérimentations ont été menées dans différentes configurations de cache (sans cache, avec cache verrouillé statiquement, avec cache non verrouillé) et ce pour différentes structures et tailles de caches.

Toutes les mesures de temps ont été effectuées en exécutant les tâches avec un jeu de données d'entrée particulier. Ce jeu d'entrée a été sélectionné pour emprunter le chemin d'exécution le plus fréquemment suivi par la tâche.

3.4 Applications de test

L'évaluation de performances a été effectuée sur 5 tâches, dont les caractéristiques sont présentées dans le tableau 1. Dans le tableau, la colonne WCET représente le WCET obtenu avec Heptane en considérant un système sans cache.

Nom	Description	Taille code (octets)	Taille données (octets)	WCET (cycles)	Localité (‰)
jfdctint	algorithme de forward DCT entière utilisée dans JPEG	3424	576	53512	8.82
crc	CRC (Codes Redondants Cycliques)	1280	1364	518974	30.29
Integral	Calcul d'intégrales par intervalles	1140	320	303279	13.61
minver	Inversion de matrices	4520	736	71139	12.57
nsichneu	Simulateur de réseau de Petri	48260	400	1067146	0.18

TAB. 1 – *Tâches de test*

La colonne *localité* (voir [PAD03] pour plus de détails) donne une indication de la localité spatiale et temporelle des accès mémoire de la tâche. Plus le facteur de localité d'une tâche est important, meilleure est sa localité tant spatiale que temporelle. Parmi les programmes considérés, *nsichneu* possède la moins bonne localité des références.

Les tâches du tableau 1 ont été groupées en deux jeux de tâches périodiques, introduits dans le tableau 2). Les périodes des tâches ont été sélectionnées pour que le système soit en surcharge si l'on utilise pas de cache (utilisation CPU calculée grâce à l'équation 1 de 130%).

Nom	Composition	Taille code (octets)
"Small"	jfdctint, crc, Integral	5844
"Big"	jfdctint, minver, nsichneu	56204

TAB. 2 – *Jeux de tâches construits à partir de tâches du tableau 1*

4 Performance du verrouillage statique de caches d'instructions

Nous comparons dans ce paragraphe les performances des deux jeux de tâche du tableau 2 avec verrouillage de cache (noté *locked* dans la suite) et sans verrouillage de cache, c'est à dire avec un cache dynamique (noté *unlocked* par la suite). Nous examinons tout d'abord l'ordonnançabilité du système (performances pire-cas) dans le paragraphe 4.1, puis ses performances mesurées (cas-moyen) dans le paragraphe 4.2. Nous examinons l'impact d'un certain nombre de paramètres (tailles des tâches, des caches, localité des références, etc) dans le paragraphe 4.3. Par défaut dans tout ce paragraphe, la séquence utilisée pour sélectionner le contenu du cache verrouillé est obtenue par exécution des tâches, et une méthode de verrouillage global est utilisée.

4.1 Ordonnançabilité (performance pire-cas, prédite)

Les résultats de l'analyse d'ordonnançabilité du système avec et sans verrouillage de cache sont donnés dans le tableau 3. Les chiffres dans le tableau représentent l'utilisation processeur pire-cas (équation 1) pour différentes tailles et degrés d'associativité du cache ; les chiffres en gras indiquent un verdict d'ordonnançabilité négatif produit par analyse de temps de réponse [3]. Pour plus de lisibilité, une représentation graphique du contenu du tableau est produite dans la figure 1.

Asso \ Taille		1Ko	2Ko	4Ko	8Ko	16Ko
1	Locked	0.55	0.42	0.33	0.33	0.33
	Unlocked	0.33	0.22	0.23	0.23	0.23
2	Locked	0.47	0.35	0.25	0.19	0.19
	Unlocked	0.42	0.33	0.23	0.23	0.23
4	Locked	0.47	0.34	0.21	0.17	0.17
	Unlocked	0.61	0.43	0.34	0.23	0.23
8	Locked	0.47	0.34	0.21	0.17	0.17
	Unlocked	0.75	0.61	0.44	0.34	0.23
16	Locked	0.46	0.34	0.21	0.17	0.17
	Unlocked	**1.00**	0.75	0.63	0.44	0.34

a. Jeu de tâches "Small"

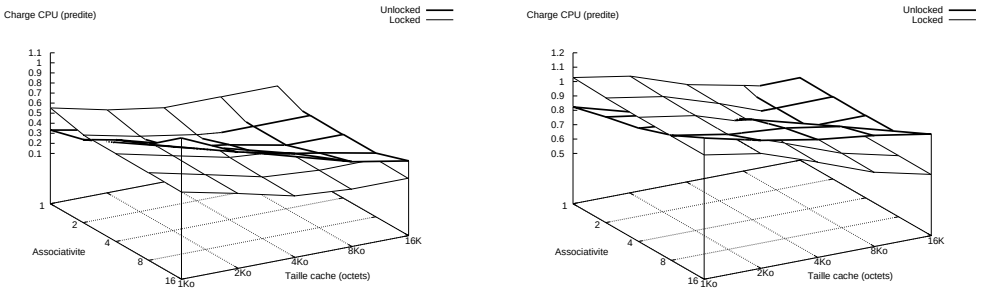
1Ko	2Ko	4Ko	8Ko	16Ko
1.02	0.96	0.82	0.74	0.65
0.82	0.70	0.71	0.71	0.72
1.01	0.95	0.82	0.67	0.57
0.88	0.82	0.71	0.71	0.72
1.01	0.94	0.82	0.66	0.57
0.93	0.89	0.84	0.72	0.72
1.01	0.94	0.81	0.66	0.57
1.01	0.95	0.91	0.85	0.72
1.01	0.94	0.81	0.66	0.57
1.13	**1.04**	0.97	0.92	0.85

b. Jeu de tâches "Big"

TAB. 3 – *Ordonnançabilité et charge CPU avec et sans verrouillage de cache*

Petits caches et faibles degrés d'associativité. Dans cette situation, l'utilisation de caches dynamique permet de prédire des performances légèrement meilleures que pour les caches verrouillés, et ce tant du point de vue de la faisabilité du système que de la charge CPU. Un élément d'explication est le fait que la méthode de verrouillage utilisée est une méthode de verrouillage global, réduisant la capacité en mémoire cache disponible

3. Un système peut respecter ses échéances malgré un verdict négatif, comme nous le verrons plus loin. En effet, la sûreté des méthodes d'analyse, en particulier pour l'obtention des WCET, peut introduire du pessimisme dans l'évaluation de l'ordonnançabilité.



a. Jeu de tâches “Small” b. Jeu de tâches ”Big”

FIG. 1 – *Utilisation processeur prédite (pire-cas) avec et sans verrouillage de cache*

pour chaque tâche. Les temps de rechargement de caches suite à une préemption sont plus importants pour les caches non verrouillés que pour les caches dynamiques. Toutefois, ce facteur a un impact très limité dans cette situation à cause de la petite taille des caches.

Impact du degré d’associativité. Pour une taille de cache donnée, on pourrait s’attendre à une amélioration des performances (i.e. baisse de la charge CPU prédite) suite à une augmentation du degré d’associativité du cache. En effet, une augmentation du degré d’associativité élimine le taux de conflits pour l’accès aux blocs de cache, que se soit au sein d’une même tâche ou entre tâches. Cette amélioration des performances prédites se produit avec les caches verrouillés, mais pas avec les caches non verrouillés. Elle ne reflète pas une baisse des performances effectives, comme nous le verrons par la suite. Elle reflète plutôt le pessimisme des méthodes de prédiction des WCET en ce qui concerne classification des instructions dans le cas de degrés d’associativité élevés.

Notons que l’algorithme de remplacement de cache utilisé ici (LRU) se prête bien à l’analyse de cache. Des résultats plus pessimistes encore auraient été produits avec des caches dont la politique de remplacement se prête moins bien à l’analyse (voir [HLTW03] pour des exemples).

Influence de la taille du cache. De manière évidente, pour un degré d’associativité donné, les performances prédites s’améliorent lorsqu’on augmente la taille du cache, que ce soit pour les caches verrouillés ou non. Ceci s’explique par une baisse du taux de conflits avec des plus gros caches. Toutefois, l’amélioration des performances est plus importante pour les caches verrouillés que pour les caches dynamiques. L’origine de cette amélioration plus importante est que les délais de rechargement de caches suite à une préemption varient linéairement avec la taille de cache pour les caches dynamiques, alors qu’ils sont constants et nuls dans le cas de caches verrouillés globalement.

4.2 Performance mesurée (cas-moyen)

L'analyse de performances pire-cas proposée dans le paragraphe précédent est importante car elle permet d'examiner l'impact de l'utilisation de caches verrouillés sur l'ordonnançabilité du système. Elle nous a permis en particulier d'identifier les situations dans lesquelles les prédictions servant à analyser l'ordonnançabilité du système sont excessivement pessimistes (degrés d'associativité des caches élevés par exemple). Toutefois, l'analyse des performances pire-cas n'est pas suffisante, car elle donne une vision volontairement pessimiste des performances du système et n'est pas représentative des performances réelles. Nous présentons donc dans ce paragraphe une analyse des performances *effectives* (mesurées) des jeux de tâches avec et sans verrouillage de caches. L'unité de mesure utilisée pour l'analyse des performances est la charge processeur mesurée du système, obtenue par exécution des jeux de tâches. Les résultats sont présentés dans le tableau 4 et sur la figure 2.

Asso \ Taille		1Ko	2Ko	4Ko	8Ko	16Ko
1	Locked	0.52	0.40	0.32	0.32	0.32
	Unlocked	0.31	0.18	0.16	0.16	0.16
2	Locked	0.42	0.33	0.24	0.18	0.18
	Unlocked	0.36	0.19	0.16	0.16	0.16
4	Locked	0.42	0.33	0.20	0.16	0.16
	Unlocked	0.36	0.19	0.16	0.16	0.16
8	Locked	0.42	0.32	0.19	0.16	0.16
	Unlocked	0.36	0.19	0.16	0.16	0.16
16	Locked	0.41	0.32	0.19	0.16	0.16
	Unlocked	0.36	0.19	0.17	0.16	0.16

a. Jeu de tâche "Small"

1Ko	2Ko	4Ko	8Ko	16Ko
0.58	0.52	0.41	0.37	0.30
0.50	0.36	0.35	0.35	0.26
0.57	0.51	0.40	0.31	0.24
0.55	0.36	0.35	0.33	0.27
0.57	0.51	0.40	0.31	0.24
0.55	0.36	0.35	0.34	0.30
0.57	0.51	0.39	0.31	0.24
0.55	0.37	0.35	0.34	0.33
0.57	0.51	0.39	0.31	0.24
0.55	0.37	0.35	0.33	0.33

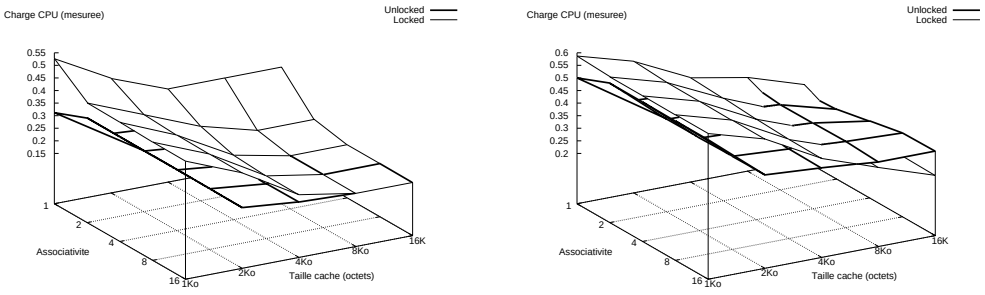
b. Jeu de tâche "Big"

TAB. 4 – *Charge CPU mesurée avec et sans verrouillage statique de cache*

Premières constatations. Une première remarque est que pour toutes les configurations de cache, la charge CPU mesurée est toujours inférieure à la charge CPU prédite, que ce soit en présence de caches verrouillés ou de caches dynamiques. Ceci confirme expérimentalement la sûreté de l'analyse d'ordonnançabilité et nous permet également d'appréhender sur ces exemples le degré de pessimisme de l'analyse d'ordonnançabilité.

Une deuxième constatation est qu'en comparaison avec un système sans cache, un système avec un cache verrouillé a de meilleures performances. La charge processeur, initialement de 130% (système infaisable), se situe entre 16% et 60% selon le jeu de tâches et la configuration de cache considérés. Bien que dans la grande majorité des cas un cache dynamique permet d'obtenir de meilleures performances qu'un cache verrouillé, un cache verrouillé, par construction, est déterministe, le prix de ce déterminisme étant des performances généralement moins bonnes.

Influence de la taille du cache. De manière évidente, pour un jeu de tâches donné et un degré d'associativité donné, une augmentation de la taille du cache entraîne une amélioration des performances mesurées, que ce soit en utilisant un cache verrouillé ou un cache dynamique. Ceci est dû à la baisse du taux de conflits pour chaque bloc de cache.



a. Jeu de tâche ”Small” b. Jeu de tâche ”Big”

FIG. 2 – *Charge CPU mesurée avec et sans verrouillage statique de cache*

Les performances mesurées sont ici totalement cohérentes avec les performances prédites. On remarque, de la même manière que pour les performances prédites, que l'amélioration des performances pour un cache dynamique est moins franche qu'avec un cache verrouillé, à cause des délais de rechargement du cache suite à une préemption.

Influence du degré d'associativité. Pour un jeu de tâches donné et une taille de cache donnée, l'augmentation du degré d'associativité du cache entraîne une amélioration des performances à la fois pour les caches dynamiques et verrouillés, à cause de la baisse du taux de conflits résultant de l'augmentation du degré d'associativité. On n'observe pas ici le moins bon comportement des caches dynamiques qui a été identifié lors de l'analyse des performances pire-cas. Cela confirme que ce mauvais comportement provient de la méthode d'analyse de cache lors de l'obtention des WCET plutôt que du degré d'associativité proprement dit.

4.3 Paramètres ayant une influence sur les performances

4.3.1 Influence de la taille des tâches

La taille du code des jeux de tâches s'avère être le facteur ayant le plus d'impact sur les performances. Nous présentons ici les variations des performances mesurées de jeux de tâches, en faisant uniquement varier le ratio entre la taille du code des jeux de tâches et la taille du cache, le cache étant un cache à correspondance directe. Pour plus de généralité, nous avons ajouté aux jeux de tâches du tableau 2 deux jeux de tâches supplémentaires, dont les caractéristiques sont données ci-dessous. Le code source de ces jeux de tâches a été généré automatiquement par un générateur de code prenant en entrée les caractéristiques du code (taux de conditionnelles, de boucles, niveau d'emboîtement

des boucles, etc).

Nom	Description	Taille code (octets)	Localité (‰)
Seq	Code quasiment séquentiel	46288	6.64
Local	Code à forte localité	43404	11.42

La figure 3 donne pour l'ensemble des jeux de tâches la variation du facteur d'amélioration Δ, représentatif de l'amélioration des performances en comparaison d'un système sans cache, en fonction du ratio *taille du code / taille du cache*. Δ est défini comme $\frac{U_{nocache}-U}{U_{nocache}}$, avec U la charge processeur mesurée. Un facteur d'amélioration Δ nul signifie qu'aucun gain en performance n'est obtenu par rapport à un système sans cache ; plus Δ est important, plus l'amélioration des performances par rapport à un système sans caches est importante.

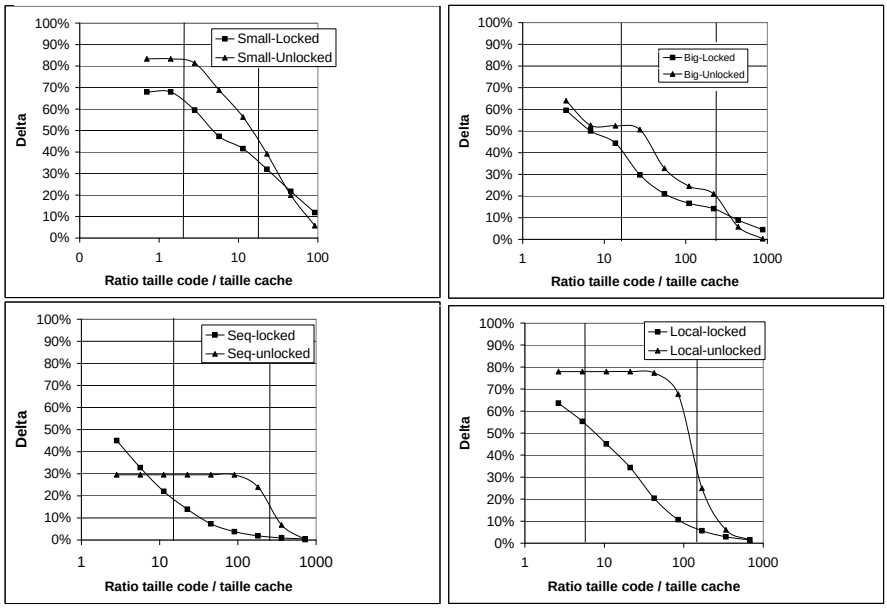


FIG. 3 – *Influence de la taille du code sur les performances mesurées*

L'examen de la figure nous permet de distinguer trois zones dans chacune des courbes, le jeu de tâches le plus représentatif des remarques formulées étant le jeu *Local* :

– La première zone correspond à des jeux de tâches, qui bien que ne logeant pas entièrement dans le cache, ne sont pas de taille sur-dimensionnée par rapport au cache (ratio allant de 4 à 30 selon le jeu de tâches). Dans cette zone, pour la plupart des jeux de tâches, la différence de performances entre caches verrouillés et caches dynamiques est plutôt faible (inférieure à 15%) et l'amélioration des performances par rapport à un système sans cache est significative.

– La seconde zone correspond à des jeux de tâches dont les tailles de code sont beaucoup plus grandes que la taille du cache (ratio variant de [4..30] à [20..300] selon les applications). Dans cette zone, la différence de performances entre caches dynamiques et verrouillés, en faveur des caches dynamiques, augmente de manière importante. La différence de performances est plus importante pour les jeux de tâches à forte localité des références, comme le jeu de tâches *Local*, pour lesquels les caches dynamiques ont un comportement optimal. Nous constatons ici que le déterminisme fourni par le verrouillage de cache a un coût en terme de performances, et que ce coût peut être très important pour les jeux de tâches de grande taille.
– La troisième zone correspond à des jeux de tâches dont la taille est démesurée par rapport à la taille du cache, et pour lesquels le taux de conflit pour des blocs de cache est très important. Dans cette zone, les performances des caches dynamiques et verrouillés s'écroulent et le facteur d'amélioration Δ tend asymptotiquement vers 0 (performances identiques à un système sans cache).

4.3.2 Influence de la localité des références des tâches

Les tâches considérées diffèrent de par leur localité spatiale et temporelle. Nous avons examiné l'impact de ce facteur sur les performances pire-cas de tâches isolées. De manière logique, les résultats obtenus, détaillés dans [PAD03], montrent que les meilleures performances sont obtenues pour les tâches possédant le meilleur facteur de localité.

4.3.3 Influence de la taille des caches : verrouillage local ou global ?

Jusqu'à présent, nous avons considéré uniquement des méthodes de verrouillage global. De telles méthodes bénéficient d'un temps de rechargement du cache nul suite à une préemption, au détriment d'une capacité dans le cache réduite pour chaque tâche. En comparaison, les méthodes de verrouillage local permettent de réduire d'avantage le temps d'exécution de chaque tâche qui utilise alors l'intégralité du cache, au détriment d'un rechargement complet du cache à chaque préemption. Ce temps de rechargement du cache est d'autant plus important que le cache est volumineux.

Si l'on note C_i^{local} (respectivement C_i^{global}) le WCET de la tâche τ_i en utilisant une méthode de verrouillage local (respectivement global), on a $C_i^{nocache} \geq C_i^{global} \geq C_i^{local}$. Concernant les temps de rechargement du cache après préemption, nous avons $\gamma_i = 0$ dans le cas d'un verrouillage global et $\gamma_i = B * t_{miss}$ pour un verrouillage local. En considérant l'utilisation CPU pire-cas $\sum_{i=1}^{N} \frac{C_i+\gamma_i}{T_i}$, le verrouillage global sera meilleur que le verrouillage local quand sa charge CPU sera moindre, c'est à dire quand :

$$\sum_{i=1}^{N} \frac{C_i^{global}}{T_i} < \sum_{i=1}^{N} \frac{C_i^{local} + B * t_{mem}}{T_i}$$

La figure 4 illustre cette analyse, en montrant la charge processeur prédite en fonction de la taille de cache utilisée.

Pour les deux jeux de tâches considérés, les performances du verrouillage local sont meilleures que celles du verrouillage global pour les petits caches. En effet, dans ce cas,

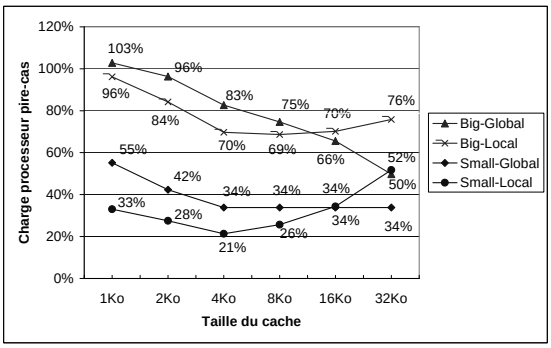


FIG. 4 – *Charge processeur pire-cas avec verrouillage local et global*

c'est la diminution des temps d'exécution des tâches, due à l'utilisation d'une méthode de verrouillage local, qui a un impact important, en comparaison avec les temps de rechargement du cache, qui sont faibles pour les petits caches. Pour les caches plus volumineux, c'est alors le temps de rechargement du cache qui a le plus d'impact, et le verrouillage global présente alors de meilleures performances.

5 Conclusion

Les mémoires caches, bien qu'améliorant le comportement moyen des programmes, introduisent des problèmes de déterminisme dans les systèmes temps-réel strict, à cause de leur comportement dynamique. Déterminer statiquement le pire comportement des programmes sur les architectures dotées de cache n'est pas toujours possible (par exemple quand la politique de remplacement du cache n'est pas assez documentée) ou peut être excessivement pessimiste. Pour contourner ce problème de déterminisme, nous avons proposé dans [PD02] l'utilisation de verrouillage statique de caches, consistant à sélectionner statiquement le contenu du cache et à le figer pendant la durée de vie du système (verrouillage global) ou d'une tâche (verrouillage local). Cet article a présenté une évaluation des performances de cet algorithme de verrouillage statique de cache.

Nous avons examiné les performances pire-cas des systèmes utilisant des caches verrouillés et les avons comparées à celles des même systèmes utilisant des caches dynamiques. Nous avons ainsi exhibé les situations dans lesquelles les méthodes d'analyse statique prenant en compte des caches dynamiques sont excessivement pessimistes (caches de grande taille, degrés d'associativité élevés). Dans ces mêmes situations, un verrouillage statique des caches se prête mieux à une analyse des performances pire-cas.

Un examen des performances mesurées a montré qu'un verrouillage statique de cache aboutit à des performances mesurées meilleures qu'avec un système sans cache, sans toutefois (en général) égaler les performances obtenues avec un cache dynamique : le déterminisme inhérent aux méthodes de verrouillage a un prix en terme de performances. Si la perte de performance induite par le verrouillage statique reste tolérable pour des jeux de tâches qui ne sont pas excessivement grands par rapport à la taille du cache, la perte en performances peut s'accroître de manière significative pour des jeux de tâches plus

volumineux. Dans ces situations, il est nécessaire d'avoir recours à des stratégies de verrouillage plus dynamiques, nécessitant des rechargements du cache en cours d'exécution des tâches. Nous travaillons actuellement sur ce type de méthodes, les grandes directions de nos travaux en cours étant présentées dans [AP03]). D'autres extensions possible à ce travail sont le traitement de caches unifiés ou multi-niveaux, ou encore la prise en compte de modèles de tâches plus complexes.

Références

[AFMW96] M. Alt, C. Ferdinand, F. Martin, and R. Wilhelm. Cache behavior prediction by abstract interpretation. In *SAS'96, Static Analysis Symposium*, volume 1145 of *Lecture Notes in Computer Science*, pages 51–66. Springer, September 1996.

[AP03] A. Arnaud and I. Puaut. Towards a predictable and high performance use of instruction caches in hard real-time systems. In *Proc. of the work-in-progress session of the 15th Euromicro Conference on Real-Time Systems*, pages 61–64, Porto, Portugal, July 2003.

[CDKM02] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Scheduling in Real-Time Systems*. John Wiley and Sons, November 2002.

[CP01] A. Colin and I. Puaut. A modular and retargetable framework for tree-based WCET analysis. In *Proc. of the 13th Euromicro Conference on Real-Time Systems*, pages 37–44, Delft, The Netherlands, June 2001.

[CPRS03] A. Colin, I. Puaut, C. Rochange, and P. Sainrat. Calcul de majorants de pire temps d'exécution : état de l'art. *Techniques et Sciences Informatiques*, 22(5):651–677, 2003.

[HLTW03] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm. The influence of processor architecture on the design and the results of wcet tools. *Proceedings of the IEEE*, July 2003. To appear.

[Kir89] D. B. Kirk. Smart (strategic memory allocation for real-time) cache design. In *Proceedings of the 10th IEEE Real-Time Systems Symposium (RTSS89)*, pages 229–237, Santa Monica, California, USA, December 1989.

[LHS+98] C. G. Lee, J. Hahn, Y. M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Transactions on Computers*, 47(6), June 1998.

[MCIBM01] A. Marti-Campoy, A. P. Ivars, and J. V. Busquets-Mataix. Static use of locking caches in multitask preemptive real-time systems. In *IEEE/IEE Real-Time Embedded Systems Workshop (Satellite of the IEEE Real-Time Systems Symposium)*, London, UK, December 2001.

[Mue00] F. Mueller. Timing analysis for instruction caches. *Real-Time Systems*, 18(2):217–247, May 2000.

[PAD03] I. Puaut, A. Arnaud, and D. Decotigy. Performance analysis of static cache locking in hard real-time multitasking systems. Technical Report 1568, IRISA, October 2003.

[PD02] I. Puaut and D. Decotigny. Low-complexity algorithms for static cache locking in multitasking hard real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS02)*, pages 114–123, Austin, Texas, December 2002.

[SS93] John E. Sasinowski and Jay K. Strosnider. A dynamic programming algorithm for cache/memory partitioning for real-time systems. *IEEE Transactions on Computers*, 42(8):997–1001, August 1993.

[TBW94] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems*, 6(1):133–151, March 1994.

[VLX03] X. Vera, B. Lisper, and J. Xue. Data caches in multitasking hard real-time systems. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS03)*, Cancun, Mexico, 2003.