UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE : Sciences et Technologie de l'Information, des Télécommunications et des Systèmes

Institut d'Electronique Fondamentale (IEF)
Thales Research & Technology - France (TRT)

DISCIPLINE : Physique

RÉSUMÉ DE LA THÈSE DE DOCTORAT EN FRANCAIS

soutenance prévue le 10/07/2014

par

Jingyi BIN

Controlling Execution Time Variability Using COTS for Safety Critical Systems

Directeur de thèse :	Alain MERIGOT	Professeur (IEF, Université Paris-sud)
Encadrants :	Sylvain GIRBAL	Ingénieur (TRT)
Composition du jury :		
Rapporteurs :	Laurent PAUTET	Professeur (LTCI, Télécom ParisTech)
	Michel AUGUIN	Directeur de Recherche (Université de Nice)
Examinateurs :	Alain MERIGOT	Professeur (IEF, Université Paris-sud)
	Claire PAGETTI	Chargée de recherche (ONERA)
	Daniel ETIEMBLE	Professeur (Université Paris-sud)
	Sylvain GIRBAL	Ingénieur (TRT)

Abstract

Au cours de la dernière décennie, le domaine safety-critical s'appuye sur les Commercial Off-The-Shelf (COTS) architectures de mono-coeur malgré leur variabilité du temps d'exécution inhérent. Aujourd'hui, l'industrie safety-critical envisage la possibilité d'utilisation des COTS de multi-coeur en tenant compte de la demande croissante de performance.

Cependant, le passage de mono-coeur à multi-coeur aggrave le problème de variabilité du temps d'exécution dû à la contention de ressources partagées. Les techniques standard pour gérer cette variabilité comme sur-approvisionnement de ressources ne peuvent pas être appliquées à multi-coeur en considérant que les safety-marges compenseront la plupart voire tout le gain de performance donné par les multi-coeurs. Une solution possible serait de capturer le comportement des mécanismes de contention potentielle sur les ressources partagées relativement à chaque application co-fonctionnant sur le système. Malheureusement, les caractéristiques sur les mécanismes de contention ne sont pas généralement clairement documentés.

Dans la thèse, nous introduisons les techniques de mesure basées sur un ensemble de stressing benchmarks et les hardware monitors à caractériser 1) l'architecture en identifiant les ressources partagées et en étudiant leur mécanisme de contention. 2) les applications en étudiant comment elles se comportent relativement aux ressources partagées. Sur la base de ces informations, nous proposons une technique à estimer le WCET d'une application dans un co-running contexte prédéterminé en simulant le pire cas des contentions sur les ressources partagées produites par co-runners de l'application.

Table des matières

Chapitre 1

Contexte

1.1 Contexte

Les systèmes safety-critical sont caractérisés par des contraintes temporelles strictes. Les applications qui tournent sur ces systèmes doivent terminer avant leur échéance. Le manquement à une échéance (une contrainte temporelle) provoquerait pour un système critique une catastrophe, p.ex. un accident d'avion. Afin de valider un système safety-critical, il faut assurer que chaque application ou tâche considérée respecte ses échéances dans tous les cas, même le pire cas. Aujourd'hui, la méthode la plus courante à garantir cette demande est d'estimer le pire temps d'exécution **(the Worst Case Execution Time (WCET))** des applications impliquées. Figure 1.1 donne les propriétés de cette estimation au niveau de la sûreté et de la précision.

En Figure 1.1, la courbe rouge, bleu et verte représente respectivement la distribution du temps d'exécution effectif, du temps d'exécution mesuré et du temps d'exécution estimé. La sûreté de l'estimation est la garantie que le WCET estimé (vert en figure) est supérieur au WCET effectif (rouge en figure). La précision de l'estimation demande que le WCET estimé doit être au plus proche de celui effectif, et la disparité entre les deux WCET est définie comme un over-margin. En conséquence, le prédictibilité de performance est plus exigé que la haute performance dans les systèmes safety-critical.

L'analyse du comportement temporel d'une application pour estimer son WCET est donc un enjeu à valider un système safety-critical. Cet analyse doit supposer le prise en compte de l'architecture sur laquelle l'application analysée s'exécute. Autrefois, afin d'assurer la sûreté de l'estimation, les systèmes

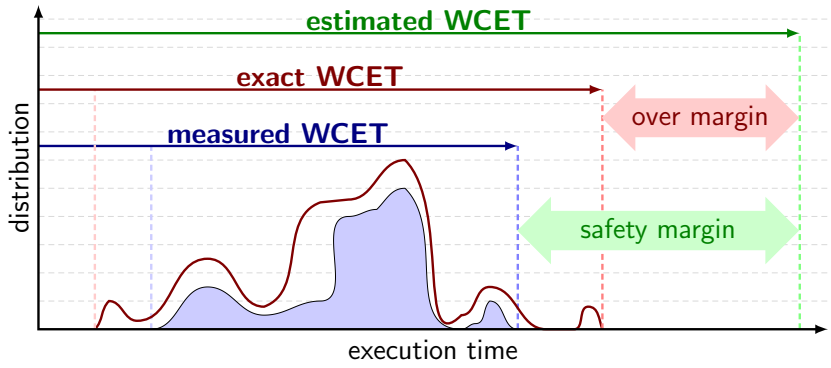

Figure 1.1 – *Estimation du WCET à satisfaire aux contraintes de sûreté et de précision*

safety-critical ont toujours utilisé les architectures mono-cœur en considérant de leur simple conception matérielle. Alors, il existe des analyses statiques et des analyses basées sur les mesures qui sont tous matures et fiables pour ce type d'architectures. Cependant, les industries safety-critical, p.ex. de type avionique, sont confrontées aujourd'hui à une demande croissante de performance, ce qui leur donne une motivation à introduire **les architectures multi-cœur**, à savoir les COTS multi-cœur qui ont le coût de NRE (Non-Recurring Engineering) plus bas et le temps vers marketing TTM (Time-to-Market) plus court en comparaison avec les in-house architectures visant à concevoir une nouvelle architecture complètement prédictible.

Les architectures multi-cœur sont composées de plusieurs cœurs d'exécution, chacun autorise une exécution en parallèle de différentes tâches. Pour ce type d'architectures, il y a certaines **ressources matérielles partagés** parmi tous les cœurs, comme le bus mémoire. Les applications qui s'exécutent en parallèle, même indépendamment dans différents cœurs, sont obligées de partager ces ressources même s'il n'y aucune communication de données entre eux. Le comportement d'une application, en présence des ressources partagées, ne dépend plus uniquement de l'application elle-même mais aussi des interactions avec celles concurrentes luttant pour les ressources communes. Ces interactions peuvent donc impacter le comportement temporel de chaque application concurrente, ce qui rend l'estimation de WCET très difficile, voire impossible. En plus de ce problème de ressources partagées, il y a un autre souci inhérent sur les COTS multi-cœur : **caractéristiques matériels non divulgués**. Certaines composantes d'un COTS ne sont pas bien documentées

dans son manuel dû à la considération commerciale, surtout les mécanismes de contention des ressources. Par exemple, pour le bus interconnect du multi-processeur P4080 développé par Freescale, il n'y a ni mécanisme de contention ni comportement de saturation ni topologie connectant des ressources off-chip. Puisque l'absence de la compréhension de mécanismes de contention de ces ressources importantes qui peuvent être les goulets d'étranglement potentiels dans le contexte de co-running applications, nous ne sommes guère en mesure de prédire la variabilité de la performance provenant de la contention sur ces ressources partagées. En plus des architectures, des co-running applications sont également la boîte noire. Nous ne savons pas comment une application autonome se comporte sur les ressources matérielles partagées, ce qui nous empêche d'étudier l'interaction entre co-running applications sur des ressources partagées.

Du coup, avant d'estimer le WCET des co-running applications, nous proposons un méthodologie caractérisant les architectures et les applications à découvrir les propriétés matérielles inconnues, surtout les mécanismes de contention des ressources, et les comportements des co-running applications relativement aux ressources partagées.

1.2 Le Plateforme - Freescale QorIQ P4080

Le P4080 est un processor de huit cœurs développé par Freescale. Comme il est un candidat potentiel du processeur de prochaine génération pour les applications avioniques, nous l'utilisons à évaluer nos approches dans la thèse. Comme démontré en Figure 1.2, le système P4080 est composé de huit cœurs de **PowerPC e500mc** couplés avec caches L1 de données+instructions privés et un cache L2 unifié privé. Une architecture de mémoire partagée dont deux caches L3 et deux contrôleurs de DDR associés est construit autour du **CoreNet fabric**. En outre, plusieurs périphériques différents sont implémentés autour du CoreNet. Nous nous concentrons sur les e500mc et le chemin de mémoire du P4080 au long de la thèse.

Compteurs de Performance En P4080, il a y 4 compteurs de performance par cœur qui sont essentiellement un groupe des registres spéciales comptant des activités matérielles du cœur, comme cycles de processeur, défauts de cache L1 .... Les compteurs de performance peuvent être accédés par une paire d'instructions *mtpmr/mfpmr*. En plus des compteurs dans les

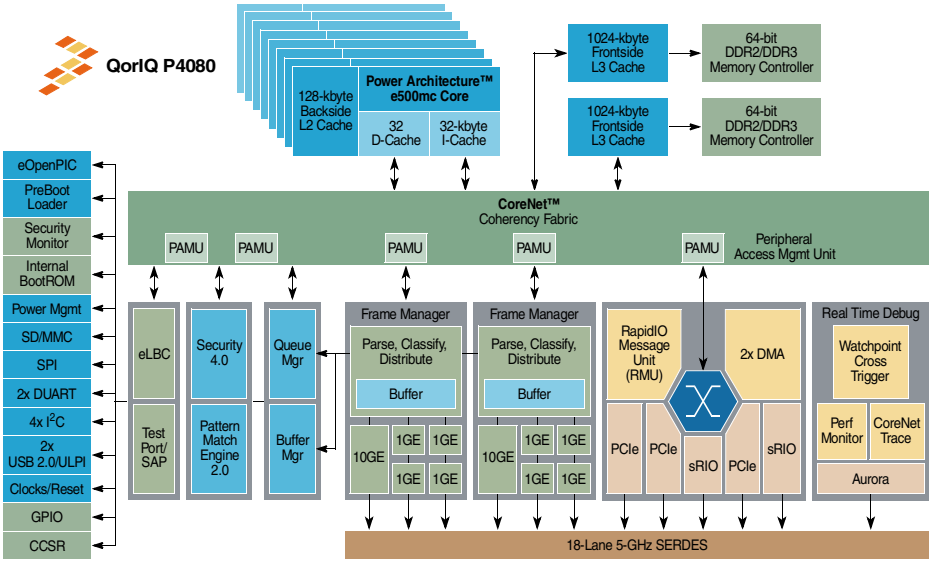

Figure 1.2 – *Diagramme du Freescale P4080*

cœurs, il existe certains compteurs au plateforme qui nous permettent de collecter des activités matérielles sur les ressources hors des cœurs, comme accès de DDR, accès de cache L3 .... Ces compteurs de performance peuvent nous aider à capturer le comportement des ressources partagées et aussi des applications au long de la caractérisation.

Chapitre 2

Quantification de la Variabilité du Temps d'Exécution

L'objective de la quantification de la variabilité est de découvrir jusqu'à quel degré le ralentissement de la performance peut atteindre à cause de la contention des ressources partagées dans un multi-cœur. Afin de évaluer la variabilité, nous profitons d'une suite des benchmarks de systèmes embarqués, dits Mibench, et de deux autres applications industrielles développées dans Thales. En considérant la facilité à porter les benchmark Mibench vers le baremetal de P4080, on a sélectionné 7 benchmarks : *ADPCM*, *CRC32*, *FFT*, *blowfish*, *SHA*, *patricia* et *susan* dans les domaines de télécommunication, de sécurité, de réseaux et d'automobile. Par rapport à ces benchmarks, les deux applications industrielles *airborne radar* et *détection de piétons* ont plus de contraintes temporelles. Bien que les 7 benchmarks ne sont pas en temps réel comme prévu pour le domaine safety-critical, ce n'est pas une question de les appliquer pour caractériser la variabilité du temps d'exécution.

Afin de quantifier la variabilité, nous concevons une suite de ressource stressing benchmarks en assembleur de sorte que nous pussions mieux maîtriser le comportement de ces benchmarks sans être impactés par la compilation. L'objective d'un ressource stressing benchmark est de produire une charge élevée sur cette ressource particulière. Nous pouvons tourner une application sous analyse indépendamment avec des stressing benchmarks stressant les ressources partagées dans différents cœurs de sorte que cet application connaîtrait un ralentissement de performance dû à la contention sur les ressources partagées stressées.

Au cours des expériences, nous avons utilisé le baremetal P4080 sans avoir

un système d'exploitation afin de minimiser l'effet de la variabilité et de faciliter l'utilisation des compteurs de performance. Lors que nous introduisons 2 stressing benchmarks avec les applications, l'impact sur le temps d'exécution moyen n'est pas significatif, cependant, pour le pire cas, on observe une variabilité moyenne de 71% et un maximum de variabilité de 361% pour le airborne radar.

En considérant que l'impact sur telle grande variabilité du temps d'exécution conduirait à des marges de WCET insoutenables loin au-dessus des avantages de performance que nous pourrions gagner des multi-cœurs, il est essentiel de contrôler cette variabilité en fournissant une caractérisation détaillée sur le mécanisme de contention des ressources matérielles partagées et comment chaque application se comporte par rapport aux ressources partagées.

Chapitre 3

Caractérisation de l'Architecture et des Applications

Pour être en mesure de contrôler la variabilité, nous avons proposé une méthodologie caractérisant l'architecture et les applications. Figure 3.1 illustre le concept de méthodologie. La caractérisation consiste en deux étapes : 1) la caractérisation de l'architecture 2) la caractérisation de l'application.

Dans la caractérisation de l'architecture, nous tournons différents ressource stressing benchmarks en parallèle pour produire et affiner une charge sur les ressources sous analyse. Pour observer la variabilité de la performance en fonction des différentes charges de ressources, nous collectons les événements en utilisant les compteurs de performance lors de l'exécution. Grâce aux stressing benchmarks et compteurs de performance, nous sommes en mesure d'identifier les ressources partagées, caractéristiques non divulgués, comme le mécanisme de contention d'une ressource partagée, et sélectionnez stressing benchmarks et compteurs de performance qui sont responsables de la variabilité du temps d'exécution.

L'objectif de la caractérisation des applications est de comprendre pourquoi le performance d'une application particulière, et donc son temps d'exécution, varie lorsque l'application tourne avec d'autres applications dans un même multi-cœur. Par conséquent, nous tournons d'abord une application avec un ensemble de ressources stressing benchmarks qui ont été sélectionnés durant la caractérisation de l'architecture afin de savoir quelles ressources sont sensible pour cet application. Deuxièmement, nous surveillons et collectons

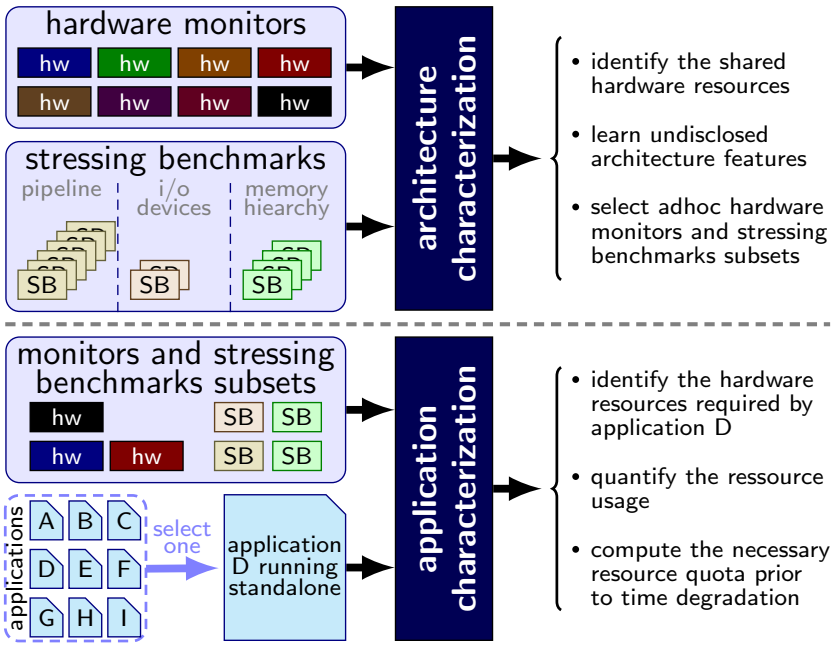

Figure 3.1 – *Processus du méthodologie*

les événements liés aux ressources partagées sensibles lors que cet application tourne en isolation, ce qui nous permet de calculer l'utilisation de chaque ressource partagée sensible. Répéter le calcul des utilisation de ressources partagées ci-dessus sur chaque application nous permettrait d'identifier applications qui pourraient fonctionner simultanément sur le système sans dépasser le débit maximum de chaque ressource partagée sensible, ce qui signifie que les pire temps d'exécution individuel ne se dégrade pas considérablement.

Selon les résultats expérimentaux, du point de vue matérielle, nous avons

- identifié des caractéristiques matérielles non divulgués : 4-core effet de groupe dans le P4080, qui a permis de réduire l'espace de conception en termes de mapping. L'application qui tente d'éviter de mapper les co-running applications dans un même groupe avec l'application sous analyse fournira la variabilité la plus faible pour cette application sous analyse.
- identifié la configuration matérielle le plus approprié pour les applications critiques. Selon deux critères : la faible variabilité et la suffisante performance moyenne, la configuration optimale est double contrôleur

/ partitionné cache L3.
– quantifié le disponibilité des ressources matérielles partagées : le débit maximal, le comportement de la saturation et de la topologie du CoreNet et du contrôleur de DDR, ce qui a permis de calculer l'utilisation des ressources dans la caractérisation des application. Soit le CoreNet ou le contrôleur DDR a une bande passante de saturation en dessous de laquelle la performance reste pratiquement inchangée. Une fois la saturation est atteinte les performances seront considérablement dégradées. Sous la configuration optimale, chaque groupe possède une bande passante basique de CoreNet, mais deux groupes partagent toujours une petite partie de la bande passante. Pour assurer que l'activité d'un groupe n'affecte pas l'activité de l'autre, nous devons faire en sorte que la saturation de la bande passante par groupe 0.219 transactions / cycle n'est pas atteint simultanément dans les deux groupes.

Du point de vue logicielle, nous avons
– obtenu le nombre d'itérations minimum pour capturer la variabilité, ce qui a permis de réduire l'espace de conception.
– capturé la sensibilité des applications aux ressources partagées.
– capturé l'utilisation moyenne et maximum des ressources.
– effectué la première prédiction sur le comportement d'une application lors qu'elle tourn avec des autres à l'aide des informations d'utilisation des ressources.

Pour les applications safety-critical, nous avons besoin d'une borne supérieure estimée du temps d'exécution des co-running applications, et de déterminer si les applications peuvent pratiquement tourner ensemble en comparant cette borne supérieure au deadline exigé. Cependant, notre première prédiction ne peut pas estimer avec précision une borne supérieure du temps d'exécution, ce qui nous empêche de garantir une exécution en toute sécurité sans mettre en danger le deadline. Afin d'estimer une borne supérieure du temps d'exécution, nous avons proposé une technique alternative pour estimer la WCET.

Chapitre 4

Technique Alternative d'Estimation du WCET

Nous proposons une méthodologie largement basés sur les techniques d'exécution / de simulation aux pire cas pour estimer facilement et rapidement le WCET d'une application en cas de co-tourner avec un ensemble prédéterminé d'applications dans les systèmes safety-critical.

La méthodologie surveille et collecte d'abord le comportement concernant les ressources partagées d'un ensemble d'applications prédéterminées tournant en isolation. Ensuite, elle identifie pour chacune de ces applications, un benchmark de remplacement qui présente un comportement borné dessus sur l'application d'origine relativement aux ressources partagées surveillés.

Tels benchmarks de remplacement devraient nous permettre de simuler le pire contention sur les ressources matérielles partagées causés par les applications d'origine. En tournant les benchmarks de remplacement avec une application sous analyse, nous serons en mesure d'estimer une borne supérieure du temps d'exécution de l'application sous analyse lors qu'elle s'exécute avec les applications d'origine.

La méthode proposée peut être appliquée dans un contexte de boîte noire sans connaître les détails complets du matériel et du logiciel. L'industrie avionique est confronté à un tel contexte lors de l'intégration des composants tiers de logiciels dans un matériel COTS avec des caractéristiques non divulgués.

Nous avons utilisé deux approches à identifier les benchmarks de remplacement : la signature globale et la signature locale.

La signature globale

La signature globale d'une application fournit des informations globales sur le comportement moyen de l'application relativement à chaque ressource matérielle. La signature globale est définie comme un vecteur de valeurs des compteurs de performance normalisées collectées au cours de l'exécution de l'application en isolation, avec les autres cœurs non utilisés. Les signatures globales nous permettent de sélectionner pour chaque application originale un benchmark de remplacement avec une signature globale la plus proche et supérieure de celle de l'application (nous avons utilisé la distance euclidienne). Ce benchmark de remplacement se comporte comme l'application originale relativement aux ressources matérielles partagées, mais avec beaucoup moins de variabilité.

Selon les expériences conçues à évaluer le résultat en utilisant la signature globale, la signature globale fournit une prédiction précise avec une marge moyenne faible à 0,72 %, mais elle ne parvient pas à systématiquement fournir les bornes supérieures, avec un taux de succès à 24,9% des cas. Cet échec peut être dû à la limitation de la signature globale qui n'est pas capable de simuler le pire contention provoquée par les pics utilisations des ressources qui sont cachée par l'effet moyen introduit dans les signatures globales.

La signature locale

Afin de surmonter la limitation de la signature globale, nous présentons une autre technique basée sur des signatures locales pour capturer le comportement de phase de l'application sur les ressources partagées, y compris les pics utilisations des ressources pouvant être utilisé à simuler le pire contention.

Par rapport aux signatures globales qui sont collectées seulement une fois au cours du plein exécution de l'application sous analyse, les signatures locales sont basées sur des techniques d'échantillonnage pour collecter l'information sur chaque intervalle du temps régulière, de nombreuses fois au cours du plein exécution de l'application. En conséquence, une signature locale n'est plus représentée par un seul vecteur de valeurs de compteurs de performance, mais par une succession de tels vecteurs. Bien que les signatures globales n'ont pu que capturer l'utilisation moyenne de chaque ressource matérielle partagée pendant toute l'exécution de l'application, les signatures locales nous permettent de capturer les pics utilisations des ressources matérielles au niveau de la fenêtre de temps en sélectionnant la valeur maximale indépendamment pour chaque compteur de performance normalisé parmi la succession de valeurs collectées. Nous pouvons alors créer un benchmark de remplacement qui a une signature globale composée de ces valeurs maximales, ce qui peut

100% capturer le pire contention sur les ressources partagées monitorées.

Parmi les évaluations réalisées en utilisant les signature locales, nous avons obtenu une marge moyenne de 12,1% et a réussi à borner le temps d'exécution dans 97,5% des cas. Par rapport à l'état de l'art, nous avons réussi à contrôler la over-marge, mais nous ne somme pas encore capable de systématiquement donner une borne supérieure, mais étant maintenant très proche.

Chapitre 5

Perspectives

Dans les expériences, la technique pour estimer le WCET a échoué dans 2,5% du nombre total de tests, ce qui peut être conclu dans une raison principale : nous nous sommes limités dans quelques compteurs de performance à représenter le comportement de l'application sur les ressources partagées, ce qui ne serait pas suffisant pour capturer le pire ralentissement de performance causés par d'autres activités non collectées. Afin de résoudre cette limitation, nous devons développer une méthode automatiquement détectant les compteurs de performance utiles qui sont responsables de la variabilité avant l'estimation. Ensuite, un benchmark de remplacement plus représentatif peut être identifié à l'aide de ces compteurs utiles pour complètement simuler le pire contention sur toutes les sources de variabilité, ce qui nous permettra de prédire le WCET dans un co-running contexte.

En plus de capacité à borner le WCET, nous envisageons de réduire la over-marge obtenue dans notre technique en tenant compte de la distribution de l'utilisation de chaque ressource et concevoir un benchmark de remplacement qui a une distribution plus pessimiste que l'application d'origine.