RenPar'15 / CFSE'3 / SympAAA'2003
La Colle sur Loup, France, 15 au 17 octobre 2003

Approche de partitionnement en ligne d'applications à temps d'exécution variable

GHAFFARI Fakhreddine *&*** ; AUGUIN Michel* ; BENJEMAA Maher** ; ABID Mohamed**

*Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis, les Algorithmes
bat. Euclide, 2000, route des Lucioles BP 121, 06903 Sophia-Antipolis Cedex
*Unité de recherche GMS. Ecole Nationale d'Ingénieurs de Sfax,
BPW 3038 Sfax Tunisie.
(Ghaffari, auguin)@i3s.unice.fr ; (Maher.Benjemaa, mohamed.abid)@enis.rnu.tn

---

Résumé
les applications de traitement d'image présentent des temps d'exécution fortement corrélés au contenu de l'image traitée [1]. L'étude du partitionnement logiciel/matériel de ces types d'applications n'est pas toujours aisé. Dans ce papier nous présentons une approche de partitionnement en ligne qui se base sur des estimations par prédictions statistiques des temps d'exécutions des tâches.

Mots-clés : partitionnement en ligne, prédictions statistiques, systèmes reconfigurables, temps d'exécution variable.

---

1. Introduction

Les systèmes enfouis font souvent appel à des architectures hétérogènes logicielles/matérielles. Afin de concevoir de tels systèmes, des méthodes de conception conjointe logiciel/matériel (codesign) sont utilisées. Ces dernières consistent à définir l'ensemble des tâches d'une application considérée et à effectuer leur répartition sur les unités logicielles ou matérielles de l'architecture.
Cette phase importante du processus de conception est le partitionnement logiciel/matériel.
A partir d'un ensemble des estimations de performances des tâches de l'application, l'outil de partitionnement vise à définir une répartition qui satisfait un ou plusieurs critères (par exemple des contraintes temporelles ou des ressources) et optimise une fonction coût (par exemple la consommation d'énergie).
De nombreuses applications, en particulier en traitement des images, ont des temps d'exécution fonction de la nature des données à traiter.
Les techniques "classiques" de partitionnement statique [2], qui déduisent une solution à partir d'une analyse des caractéristiques (temps d'exécution, coûts) des fonctions de l'application exprimées sous forme de constantes ne sont pas adaptées. En effet, elles conduisent à maximiser systématiquement les temps d'exécution et le nombre de ressources dans l'architecture avec pour conséquence un pessimisme excessif qui conduit à sur-dimensionner les architectures produites.
Classiquement le partitionnement d'une application nécessite au préalable de déterminer avec précision les caractéristiques des fonctions de l'application. L'évaluation des temps d'exécution dans le pire cas (WCET: Worst Case Execution Time) est nécessaire pour déduire une implémentation temps réel. Or dans le cas du traitement d'images vidéo il peut suffire d'optimiser la qualité de service dans de nombreuses applications ce qui autorise alors des dépassements éventuels d'échéances temporelles. Dans [3] nous avons proposé une approche de partitionnement basée sur des graphes de flots de données conditionnées pour considérer une caractérisation suivant des WCET par intervalles de temps d'exécution des tâches.
Les applications en relation à la vidéo nécessitent généralement des volumes de calcul importants à réaliser entre deux images successives mais le flux vidéo est relativement lent (typiquement 40 ms entre deux images). Aussi il est envisageable dans ce cas de considérer un partitionnement effectué en ligne et donc en temps réel qui opère sur des estimations des temps d'exécution issues de mesures sur les tâches exécutées sur les images précédentes pour produire une allocation utilisée à l'image suivante. L'hypothèse

réaliste effectuée est que les objets dans une séquence de quelques images fluctuent lentement. La mise en œuvre d'un partitionnement dynamique nécessite donc d'effectuer des estimations sur la base de mesures de temps d'exécution et de développer un algorithme de partitionnement qui réalise un compromis entre efficacité et vitesse d'exécution.
Ce dernier point associé à l'objectif d'éviter d'introduire des perturbations sur les traitements des tâches militent pour considérer une architecture avec une unité matérielle reconfigurable dynamiquement. L'intérêt de cette technologie se généralise avec des composants à faible coût et de grande taille (plusieurs millions de portes) ce qui conduit plusieurs équipes à étudier des architectures à base de FPGA, par exemple [4], [5].
La suite de ce papier est organisée comme suit : dans le deuxième paragraphe, nous détaillons notre approche de partitionnement en ligne, ensuite nous présentons une méthode d'estimation du temps d'exécution dans le paragraphe 3. L'algorithme de partitionnement est présenté dans un quatrième paragraphe suivi par une description de l'architecture cible pour finir par nos conclusions et perspectives.

2. Approche de partitionnement en ligne

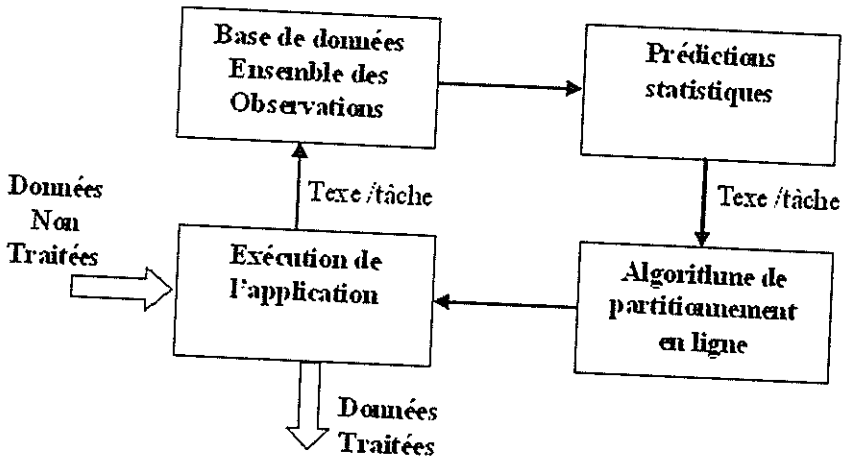

Figure 1: Approche de partitionnement en ligne

Notre approche du partitionnement logiciel/ matériel peut se résumer par la figure 1.
L'algorithme de prédiction statistique prend en entrée les observations (mesures du temps d'exécution sur les images précédentes) et fournit en sortie une estimation du temps d'exécution des tâches pour la prochaine itération de l'application. L'algorithme de partitionnement en ligne prend à chaque itération ces estimations du temps d'exécution et donne en sortie un ordonnancement et un schéma d'allocation de toutes les tâches sur les unités de l'architecture cible pour l'itération suivante.
A l'issue de chaque itération, l'ensemble des mesures des tâches de l'itération i réactualise la base de données qui contient l'ensemble des anciennes observations.

3. Estimation des temps d'exécution

Dans la littérature, il existe trois approches d'estimations des temps d'exécution : la première [6] consiste à analyser le code source de la tâche pour en déduire un WCET relativement à une architecture donnée. Cette méthode convient pour effectuer un partitionnement hors ligne mais avec l'inconvénient du pessimisme évoqué ci-dessus.
La deuxième [7,8,9] se base sur une analyse du code de l'application qui détermine sa composition en terme des primitives pré-caractérisées auparavant. Ces primitives sont des codes types (benchmarks), pour lesquels des mesures de temps d'exécution sont déjà effectuées.

La dernière approche [10,11,12] est basée sur des techniques de prédiction statistique qui utilisent les observations du passé pour prédire les temps d'exécution. C'est ce type d'approche qui est considéré dans nos travaux.
La figure 2[13] montre un exemple de densité de probabilité des temps d'exécutions d'une tâche de calcul matriciel. Nous remarquons que le pire des cas (WCET) est très peu probable (probabilité proche de zéro) et que majoritairement le temps d'exécution est très inférieur au WCET.

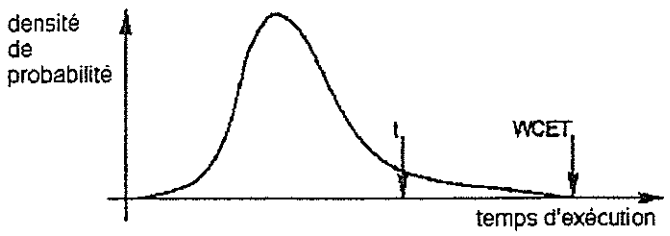

Figure 2: Densité de probabilité du temps d'exécution d'une tâche

Sur une architecture reconfigurable le temps d'exécution d'une tâche dépend également du nombre de ressources affectées à l'exécution de cette tâche. Par exemple sur la figure 3, un calcul de WCET est réalisé pour différentes quantités de ressources d'un FPGA qui permettent d'exploiter à des degrés divers le parallélisme de la tâche. A l'exécution, pour chaque quantité de ressources considérées, nous obtenons des temps d'exécution inférieurs à cet ensemble de WCET (figure 4).
On conçoit aisément que le partitionnement logiciel/matériel devrait conduire à des solutions différentes suivant que l'on considère les WCETs de la figure 3 ou les temps effectifs de la figure 4.

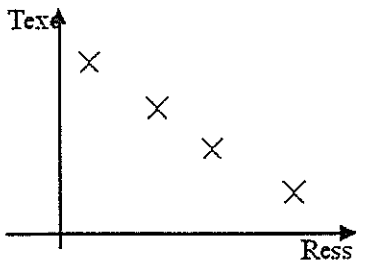

Figure 3: Courbe d'implantations avec les WCETs

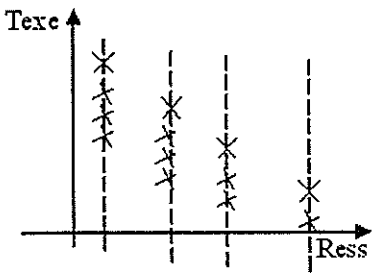

Figure 4: Courbe d'implantations avec les estimations des temps d'exécutions

Dans la suite, nous ne considérons pas tous les temps d'exécution possibles, car cela nécessitera d'effectuer une synthèse en ligne des tâches ce qui est inenvisageable. Nous proposons de sélectionner quelques points comme illustré dans [3] pour lesquels ce travail de synthèse est réalisé hors ligne.
La méthode d'estimation envisagée est inspirée de celle de Iverson [14]. Pour chaque tâche, les temps d'exécution sont mesurés en fonction d'un paramètre de corrélation, ensuite introduits dans la base des données (voir figure 5). Le paramètre de corrélation correspond à la variable responsable de la fluctuation du temps d'exécution d'une tâche en fonction de la nature des images [3].

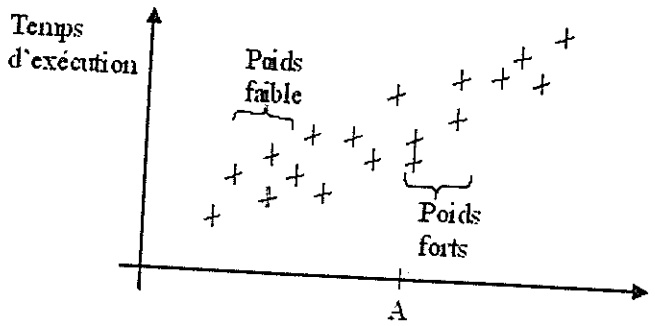

Figure 5: Méthode de K plus proches voisins (K-ppv)

A chaque itération le temps d'exécution de chaque tâche est estimé par la formule suivante (1):

$$T = m(x) + e$$

Où T est l'estimation du temps d'exécution, m(x) est le temps d'exécution qui dépend du paramètre de corrélation x (extrait du courbe de la figure 5) et e est un taux d'erreur (paramètre qui ne dépend pas de x).
La valeur de m(x) est calculée par la formule suivante (2).

$$m(x) = \frac{1}{n}\sum_{i=1}^{n} W_i(x)t_i$$

où n est le nombre de mesures disponibles, les coefficients $w_i$sont les poids affectés aux mesures et les $t_i$sont les valeurs des mesures.
Comme le montre la figure 5, en fixant une valeur au paramètre de corrélation (à A par exemple sur la figure 5), la valeur de l'estimation m(x) est obtenue par la somme pondérée à partir des voisins, la pondération étant fonction de la position de ce paramètre de corrélation. Des poids forts sont affectés aux voisins les plus proches et des poids de plus en plus faibles aux voisins les plus éloignés.
Pour des raisons de simplicité, nous calculons uniquement la variance de l'erreur e par la formule suivante (3):

$$\sigma^2(e) = \frac{1}{n}\sum_{i=1}^{n} W_i(x)(t_i - m(x))^2$$

A partir du temps d'exécution ainsi estimé, nous considérons le temps d'exécution supérieur ou égal à celui qui a été utilisé pour effectuer la synthèse hors ligne de la tâche.

4. Algorithme de partitionnement en ligne

Le comportement d'une application de traitement d'image peut être décrit par un graphe de flot de données où chaque opération de traitement est modélisée par un nœud du graphe et les arcs représentent les dépendances de données entre les tâches.

Une étude bibliographique a été entreprise pour la recherche d'un algorithme d'allocation/ ordonnancement qui réalise un compromis complexité/efficacité. L'algorithme devrait s'adapter à une implémentation sur FPGA afin d'assurer la rapidité et la flexibilité.
Plusieurs approches d'ordonnancements sur des cibles multiprocesseurs tiennent en compte de la communication inter-processeurs par exemple dans [17,18,19,20]. Mais très peu des approches qui ciblent des architectures hétérogènes [21,22]. Deux algorithmes d'ordonnancement ont été proposés par Wu et Gajski dans [20]: l'algorithme MCP (Most critical Path algorithm) et l'algorithme MD (Mobility Directed algorithm). La différence entre ces deux algorithmes est la méthode de sélection de la tâche qui sera prête à l'exécution.
Pour notre cas, nous pouvons par exemple avoir recours à l'algorithme HCP (Heterogeneous Critical Path) élaboré par Madsen [15] qui constitue une approche d'ordonnancement sur une architecture multiprocesseurs hétérogène tenant compte de la communication entre les unités de traitement. Cet algorithme commence par le calcul des priorités pour chaque tâche associée à un processeur. La tâche est choisie en fonction de la longueur de son chemin critique (CPL : Critical Path Length). La tâche qui a le plus grand minimum des CPL est la plus prioritaire. Une fois cette tâche sélectionnée, elle est affectée à l'unité de traitement qui vérifie l'ordonnancement au plutôt en tenant compte de la communication avec ses prédécesseurs.

5. Architecture cible

Pour tenir compte des résultats en temps réel du partitionnement, une architecture composée d'un (ou de plusieurs) processeur(s) connecté(s) à une unité reconfigurable dynamiquement semble judicieuse.
Cependant une architecture capable de supporter une reconfiguration avec un ordonnancement en ligne pose des difficultés certaines en particulier sur les communications et l'ordonnancement des traitements du fait qu'ils dépendent des choix effectués par le partitionnement. Des études complémentaires sur l'architecture sont nécessaires pour prendre en compte l'ensemble de ces aspects. Une approche consiste à utiliser l'interface générique proposé dans [16] qui intègre un contrôleur dont le but est de "virtualiser" le matériel afin de lui associer une flexibilité équivalente au logiciel.
Pour un algorithme de partitionnement en ligne, il faut que le temps nécessaire au partitionnement plus le temps mis par l'algorithme d'estimation des temps d'exécution soit inférieur au temps d'exécution de l'application à partitionner, c'est-à-dire qu'à chaque itération il faut garantir un nouveau résultat de partitionnement avec les nouvelles estimations des temps d'exécutions. Ceci nous a conduit à allouer une unité reconfigurable pour l'algorithme de prédiction et l'algorithme de partitionnement en ligne. En effet, il faut pour ces deux algorithmes assurer d'une part que leur temps cumulé de calcul soit inférieur au plus petit temps d'exécution d'une itération de l'application et d'autre part avoir une flexibilité de programmation pour tenir compte d'évolutions ou de modification dans la description sous forme de tâche de l'application. Pour l'exécution de l'application, nous ciblons une architecture constituée par un processeur connecté à une unité reconfigurable qui permet des reconfigurations partielles. Une mémoire RAM est également ajoutée pour mémoriser les observations des temps d'exécutions et contenir les tables issues du partitionnement à destination de l'ordonnanceur, vraisemblablement mis en œuvre sur le processeur. Une représentation de cette architecture est donnée par la figure 6.

6. Conclusion

Nous avons présenté dans ce papier une approche de partitionnement logiciel/matériel en ligne qui utilise un algorithme de prédiction statistique. Pour chaque itération, ce dernier fournit un temps d'exécution à l'algorithme de partitionnement pour chercher l'allocation/ordonnancement adéquat. Nous comptons implanter les deux algorithmes (estimation et partitionnement) sur une unité reconfigurable, et l'application sur un processeur connecté à un FPGA. Le choix de ces deux algorithmes ainsi que l'architecture cible doit être prouvé expérimentalement. Actuellement, nous essayons d'expérimenter notre approche de partitionnement sur une application de traitement d'image qui consiste à faire la détection de mouvement sur un fond d'image fixe.
Il s'agit d'un projet ambitieux et novateur qui soulève de nombreuses questions. Les solutions architecturales sont envisageables mais elles restent à être précisées. De plus, l'efficacité d'une telle approche par

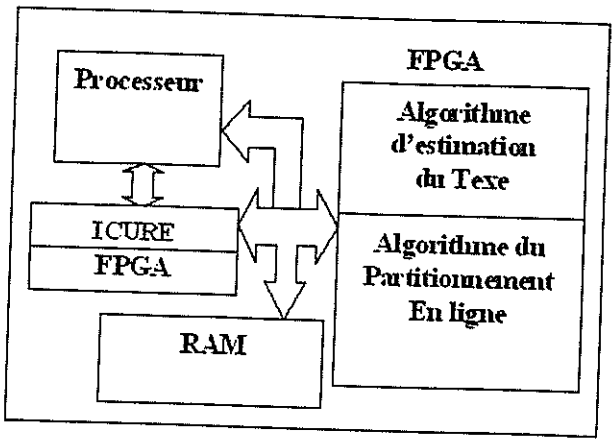

Figure 6: Une représentation de l'architecture cible

rapport à des solutions traditionnelles doit être vérifiée.

Bibliographie

1. F.GHAFFARI, Etude du partitionnement logiciel/matériel d'applications à distribution variable de charge de calcul, Rapport de stage de DEA de l'ENIS effectué à I3S université de Nice Sophia-Antipolis, 2001/2002.

2. B. Dave, N. Jha, CASPER, concurrent hardware-software co-synthesis of hard real time aperiodic and periodic specifications of embedded system architectures, DATE'98, Paris, pp 118-124, 1998.

3. F.GHAFFARI, M.AUGUIN, M.BENJEMAA. Etude du partitionnement logiciel/matériel d'applications à distribution variable de charge de calcul. Renpar'14 /ASF/SYMPA pp. 334-338 Hammamet, TUNISIE 10 – 13 avril 2002.

4. J.M. Saul, Hardware/Software Codesign for FPGA-Based Systems, Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences, 1998.

5. J. R. Hauser, J. Wawrzynek, Garp: A MIPS Processor with a Reconfigurable Coprocessor", IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '97, April 16-18, 1997.

6. B. Reistad and D. K. Gifford. Static dependent costs for estimating execution time. In *Proc. of the 1994 ACMConference on LISP and functional programming*, pages 65–78. ACM Press, June 1994.

7. H. J. Siegel. Heterogeneous computing. *Annual Research Summary 5.92, http://ece.www.ecn.purdue.edu/Researchsummary/Section5/sec5 92.html*, 1994.

8. J. Yang, I. Ahmad, and A. Ghafoor. Estimation of execution times on heterogeneous supercomputer architectures. In *the 1993 Inter. Conf. on Parallel Processing*, volume 1, pages 219–226. CRC Press, Aug. 1993.

9. T. Yang and A. Gerasoulis. DSC: Scheduling tasks on an unbounded number of processors. *IEEE Trans. Parallel and Distributed Systems*, 5(6):951–967, Sept. 1994.

10. M. V. Devarakonda and R. K. Iyer. Predictability of process resource usage: A measurement-based study on UNIX. *IEEE Trans. Software Engineering*, 15(12):1579–1586, Dec. 1989.

11. M. A. Iverson, F. ¨ Ozg¨uner, and G. Follen. Run-time statistical estimation of task execution times for heterogeneous distributed computing. In *Proc. of the 1996 High Performance Distributed Computing Conference*, pages 263–270, Syracuse, NY, Aug. 1996.

12. T. Kidd, D. Hensgen, L. Moore, R. Freund, D. Charley, M. Halderman, and M. Janakiraman. Studies in the useful predictability of programs in a distributed and homogeneous environment. *The Smartnet Home Page (http://papaya.nosc.mil:80/SmartNet/)*, 1995.

13. Sorin Manolache *"Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times"*,
Licentiate thesis no. 985, Linköping University novembre 2002.

14. Michael A. Iverson, Füsun Özgüner, Lee C. Potter, Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment, Eighth Heterogeneous Computing Workshop April 12 - 12, 1999 San Juan, Puerto Rico

15. P. Bjorn-Jorgensen, J. Madsen , Critical path driven cosynthesis for heterogeneous target architectures 5th International Workshop on Hardware/Software Co-Design (Codes/CASHE '97) March 24 - 26, 1997.

16. M.Auguin et al, EPICURE : A partitioning and CoDesign Framework For Reconfigurable Computing, soumis à CODES_ISSS'03.

17. T. Adam, K. Chandy, and J. Dickson. A comparison of list scheduling for parallel processing systems. Comm. ACM, 17(12):685–690, December 1974.

18. J. Hwang, Y. Chow, F. Anger, and C. Lee. Scheduling precedence graphs in systems with interprocessor communication times. SIAM J. Computing, 18(2):244–257, April 1989.

19. Y. Kwok and I. Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. IEEE Trans. Parallel Distrib. Syst., 7(5):506–521, May 1996.

20. M. Wu and D. Gajski. Hypertool: A programming aid for message-passing systems. IEEE Trans. Parallel Distrib. Syst., 1(3):330–343, July 1990.

21. G. Sih and E. Lee. A compile-time scheduling heuristic for itercennection-constrained heterogeneous processor architectures. IEEE Trans. Parallel Distrib. Syst., 4(2):175–187, February 1993.

22. T.-Y. Yen. Hardware-Software Co-Synthesis of Distributed Embedded Systems. PhD thesis, Princeton University, 1996.