Une méthodologie descendante pour la conception de systèmes matériels/logiciels

Pascal Gomez
Équipe ASIM/LIP6
Université Pierre et Marie Curie
Paris

E-mail: `Pascal.Gomez@lip6.fr`

Résumé

Cet article présente la méthodologie de conception conjointe matériel/logiciel des systèmes associée à l'ensemble d'outils Disydent. C'est une méthode qui laisse certaines phases de la conception, comme le partitionnement en tâches, le choix de l'implémentation, matérielle ou logicielle, de chaque tâche, entièrement à la charge du concepteur. Cette méthode est illustrée par la conception d'un décodeur de flux d'images Jpeg.

1. Introduction

Les méthodologies de conception conjointes matériels/logiciels récentes suivent généralement une approche descendante dont les principales étapes sont :

spécification La spécification du système est écrite dans un langage de haut niveau (VHDL, ...) comme un ensemble de tâches communicantes. La communication se fait grâce à des primitives bloquantes prédéfinies.

validation et évaluation La validation fonctionnelle se fait par simulation de la spécification. Des modèles de coût des composants matériels sont utilisés pour l'évaluation temporelle du système. Le choix du type d'implémentation de chaque tâche est déterminé par les contraintes du système et par les résultats de l'évaluation temporelle.

synthèse logiciel/matériel La partie logicielle est compilée sur le CPU cible et lié à une implémentation des primitives de communications. La synthèse matérielle de haut niveau permet d'obtenir les composants matériels du système.

Disydent propose une méthode basée sur l'utilisation d'une spécification initiale en C, reposant sur une sémantique spécifique pour les entrées/sorties permettant de faire tardivement le choix d'implémentation des tâches.

2. Méthodologie

La méthodologie de conception utilisée suit une approche descendante :

– écriture d'une spécification multi-*threads* entièrement logicielle ;
– validation fonctionnelle de la spécification. L'évaluation temporelle est obtenue par l'utilisation d'un simulateur *cycle true*.
– évaluation et validation de plusieurs découpages logiciels/matériels.

3. Spécification multi-*threads*, graphe de tâches

La spécification initiale est décrite sous forme d'un ensemble de tâches communicantes. Il s'agit d'une spécification entièrement logicielle, en C, utilisant les fonctions POSIX. Chaque tâche est décrite par un *thread*, communicant avec les autres *threads* à travers des fifos. On peut représenter cette spécification par un graphe de tâches (voir figure 1), sur cette figure les noeuds représentent les tâches, les arcs représentent les communications (fifos).

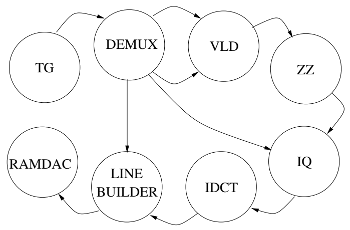


FIG. **1. Graphes de tâches pour le MJpeg. La tâche TG reçoit le flux de données compressées, la tâche Ramdac affiche l'image décompressée.**

Les fifos sont dans ce cas purement logicielles : il s'agit de *buffers* dont l'accès est protégé par des mutex, qui rendent les communications bloquantes. Les accès sont gérés par deux primitives de communications : *void* **fifoRead**(*channel, buffer, nbytes*) et *void* **fifoWrite**(*channel, buffer, nbytes*). La validation fonctionnelle se fait par compilation et éxécution de la spécification sous Solaris ou Linux.

4. Environnement de simulation

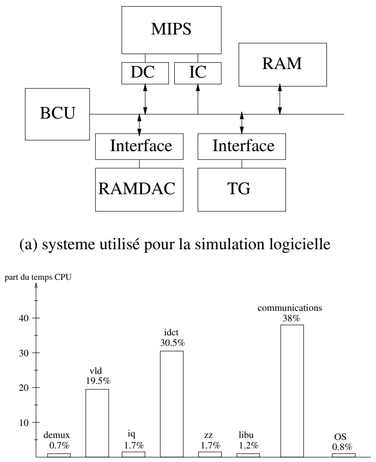
Nous utilisons CASS qui est un simulateur *cycle true* (*Cycle Accurate System Simulator*), l'utilisateur doit fournir :

– une description C des processeurs et des modules matériels ;

– une netlist VHDL décrivant l'interconnexion entre les composants possédant un modèle C.

L'utilisateur peut, grâce à CASS, charger le programme dans une RAM et simuler le système. Les résultats obtenus sont : des informations de *profiling* du code, le nombre de cycles utilisés ainsi que les taux de *miss* des caches.
Pour la simulation il faut disposer des modèles CASS de chacun des modules de l'architecture cible, généralement un bus sur lequel sont connectés un ou plusieurs processeurs, une RAM, et des coprocesseurs. Les interfaces des coprocesseurs sont de type fifos, ce qui permet notament de les connecter directement entre eux. Pour les connecter à un *bus*, on utilise un module de service qui sert d'interface entre le *bus* et le coprocesseur.

L'architecture initiale de l'exemple MJpeg (figure 2 (a)) est composé d'un Pi-Bus sur lequel sont connectés un processeur MIPS r3000, par l'intermédiaire d'un cache instruction et d'un cache de données, d'une RAM et de deux coprocesseurs qui effectuent la lecture des données (fichier .jpg) et l'affichage du résultat (image décompressée). Ce système sert à éxécuter, sous CASS, l'application multi-*threads* initiale, ce qui nécessite le développement d'une implémentation des *threads* POSIX pour le MIPS r3000.



(a) systeme utilisé pour la simulation logicielle

(b) profiling

FIG. **2. (a) Système minimal utilisé pour la simulation d'un ensemble de tâches logicielles. (b) Temps CPU utilisé pour chacune des tâches logicielles.**

5. Partitionnement logiciel/materiel

Le partitionnement logiciel/matériel est entièrement laissé à la charge du concepteur, nous voulons néanmoins un moyen rapide d'évaluer une architecture.

5.1 Threader

Le *threader* est un modèle CASS effectuant le transfert de données entre une interface fifo materielle (modèle CASS) et un canal de communication logiciel. Ce mécanisme permet d'utiliser un ou plusieurs *threads* logiciels pour simuler le fonctionnement d'un coprocesseur matériel : les *threads* sont connectés au *threader* par des canaux de communications logiciels. L'ensemble est vu par le simulateur comme un modèle materiel permettant une simulation *cycle precise*. Le *threader* gère des effets materiels, comme le débit ou la latence, qui permettent de s'approcher du fonctionnement réel d'un coprocesseur matériel.

5.2 Communications

Les communications se font entre des tâches qui peuvent être materielles ou logicielles, il faut donc disposer d'un ensemble de primitives de communications entre tâches, qui recouvre tous les schémas possibles : logiciel-logiciel, matériel-logiciel, logiciel-matériel et matériel-matériel, plusieurs stratégies étant possibles pour chacun de ces schémas. Le code d'une tâche doit être indépendant du schéma de communication utilisé (pas de synthèse logicielle), pour l'utilisateur seule la façon d'initialiser le canal de communication est différente entre deux schémas de communications.

Les primitives de communications et le *threader* permettent, pour la simulation, de passer rapidement d'une implémentation logicielle à une implémentation matérielle d'une tâche et vont faciliter l'exploration architecturale.

6. Exemple : MJpeg

La spécification du décodeur MJpeg est un ensemble de huit tâches communicantes. Pour les simulations, nous utilisons un flux JFIF avec une taille d'image de 320x200 pixels. La simulation sous forme d'un système entièrement logiciel, sous CASS, permet d'obtenir des informations de *profiling* logiciel (Voir figure 2 (b)). Ces résultats montrent que les tâches VLD et IDCT utilisent 50 % du temps CPU, alors que les autres tâches en utilisent chacune moins de 1,5 %. D'autre part les communications entres tâches logicielle utilisent près de 40 % du temps CPU. L'utilsation de tâches matérielles réduit le temps CPU utilisé pour les tâches mais aussi celui utilisé par les communications logicielles/logicielles, les communications materielles/logicielles étant beaucoup moins coûteuses. La contrainte temporelle imposée est d'afficher 25 images par secondes. Le simulateur donne le nombre de cycles utilisés lors de la simulation, on en déduit la fréquence à laquelle devra fonctionner le système pour afficher 25 images par secondes. Pour un décodeur avec uniquement des tâches logicielles, cette fréquence est de 1,2 GHz. Elle tombe à 613MHz si la seule tâche IDCT est implémentée en matériel. Pour une implémentation de toutes le tâches en matériel, sauf la tâche DEMUX, la fréquence est de 16.5 MHz. Ces implémentations matérielles des tâches ont été testés rapidement grâce au threader.

7. Conclusion

L'expérimentation sur le décodeur MJpeg permet de valider la méthodologie, et montre qu'elle permet une exploration architecturale rapide grâce au threader et au simulateur CASS, qui effectue une simulation à 100K cycles par secondes. La suite de ce travail est la synthèse des tâches matérielles avec l'outil de synthèse de haut niveau UGH.