TP2: Introduction d'un coprocesseur matériel spécialisé

TP Précédent: MjpegCourse/Multipipe

0. Objectif

L'objectif de ce TP est de vous montrer comment introduire un coprocesseur matériel spécialisé dans une architecture matérielle comportant principalement des processeurs programmables.

L'introduction d'un accélérateur matériel n'est pas toujours justifiée. La loi d'Amdhal précise qu'il est inefficace d'accélérer un traitement qui ne représente qu'une petite partie du temps de calcul total de l'application. De plus, il faut prendre en compte les temps de communication entre le coprocesseur et le reste de l'application.

La tâche `idct` étant la plus gourmande en temps de calcul, nous allons analyser les gains de performance apportés par l'implantation de la tâche `idct` comme un processeur matériel spécialisé.

Pour éviter de re-écrire toute l'application logicielle, on ne veut pas modifier la structure du TCG. Par conséquent, le coprocesseur matériel doit utilise les mêmes canaux de communication MWMR (en entrée et en sortie), que ceux utilisés par la tâche logicielle. Il faut donc que le coprocesseur matériel respecte le protocole MWMR à 5 étapes:

- prise du verrou,
- consultation de l'état de la FIFO logicielle,
- transfert des données,
- mise à jour de l'état de la FIFO logicielle,
- libération du verrou.

Pour simplifier le travail des outils de synthèse, et séparer clairement les fonctions de calcul et les fonctions de communication, ce n'est pas le coprocesseur matériel synthétisé qui implémente le protocole MWMR. On utilise pour accéder aux canaux MWMR un composant matériel générique, appelé contrôleur MWMR. Cet initiateur VCI est capable de lire ou d'écrire dans les canaux MWMR (en respectant le protocole à 5 étapes), et fournit au coprocesseur autant d'interfaces de type FIFO que celui-ci en a besoin. Ce composant est également une cible VCI, puisqu'il doit être configuré par le logiciel. C'est ce même contrôleur MWMR qui a déjà été utilisé pour interfacer les composants matériels RAMDAC et TG. Nous repartirons de la plateforme du TP3: VgmnNoirqMulti. Nous modifierons cette plateforme comportant trois processeurs Mips, pour remplacer un des processeurs programmable par un coprocesseur matériel dédié à la transformation IDCT.

Mettre ici le dessin de la plate-forme matérielle complête avec 2 processeurs et 3 controleurs MWMR

Reprenez les fichiers du TP3:

- La description de la plateforme matérielle
- La description de l'application (c'est à dire le TCG et les directives de déploiement)

- Le code des tâches (`Libu` ne gère qu'un seul pipeline et `Split` n'existe pas)

? Q1. Rappelez le temps nécessaire pour décoder 25 images, dans le cas d'une implantation utilisant trois processeurs, lorsque la tâche `idct` est placée sur un processeur, que la tâche `vld` est placée sur un second processeur, et que toutes les autres tâches logicielles se partagent le troisième processeur.

1. Coprocesseur virtuel

Il existe plusieurs solutions micro-architecturales pour la réalisation d'un coprocesseur matériel spécialisé. Dans le cas d'une transformation IDCT, on peut, suivant le nombre d'opérateurs arithmétiques utilisés, effectuer le calcul d'un bloc de 64 pixels en un cycle, en 8 cycles, en 64 cycles, en 512 cycles ou plus. En première approxiation, le coût matériel est inversement proportionnel au temps de calcul.

Pour éviter de gaspiller du silicium, il faut donc - avant de se lancer dans la synthèse - évaluer précisément la puissance de calcul requise pour le coprocesseur, une fois celui-ci placé dans son environnement de travail. Il faut donc faire de l'exploration architecturale *avant synthèse*.

Pour cela, on commence par *émuler* le coprocesseur matériel - sans le synthétiser - en encapsulant la tâche logicielle `idct` existante dans un composant matériel générique appelé *threader*, qui est un service fourni par l'environnement DSX. Pour ce qui concerne le matériel, ce composant *threader* s'interface avec le composant matériel *contrôleur MWMR*, mais il est également capable de communiquer avec la tâche logicielle `idct`, de façon a utiliser ce code pour effectuer les calculs qui devront être réalisés par le coprocesseur matériel. En pratique, la simulation dans ce mode consiste à exécuter un programme parallèle comportant deux processus UNIX communicant entre eux par des *pipes* UNIX.

- Le premier processus est le simulateur SystemC modélisant l'architecture matérielle (y compris le *contrôleur MWMR* et le composant *threader*).
- Le second processus est la tâche logicielle `idct` encapsulée dans le composant *threader*.

Pour utiliser ce mode d'émulation, il faut modifier deux choses dans la description DSX:

- dans la définition du modèle de la tâche `idct`, il faut ajouter l'implémentation `SyntheticTask()`

```
idct = TaskModel(
                'idct',
                infifos = [ 'input' ],
                outfifos = [ 'output' ],
                impl = [ SwTask( 'idct',
                                                    stack_size = 1024,
                                                    sources = [ 'src/idct.c' ],
                                                    defines = [ 'WIDTH', 'HEIGHT' ] ),
                                        Synthetic()
                                        ] )
```

- Dans la partie déploiement, il faut déployer la tâche `idct` comme une tâche matérielle (comme on l'a fait pour les tâches `ramdac` ou `tg`.

```
mapper.map("idct", vci = mapper.hard.vgmn)
```

Le coprocesseur matériel IDCT (comme beaucoup de coprocesseurs matériels de type *flot de données*) exécute une boucle infinie dans laquelle il effectue successivement les actions suivantes:

1. recopie d'un bloc de 64 coefficients du canal MWMR d'entrée vers une mémoire locale BUFIN,
2. calcul d'un bloc de 64 pixels, et stockage de ces pixels dans une seconde mémoire locale BUFOUT,
3. recopie de ces 64 pixels de la mémoire locale BUFOUT vers le canal MWMR de sortie.

Compte-tenu des caractéristiques Pour modéliser le temps de traitement la tâche matérielle virtuelle plus exacte en temps de simulation, on peut ajouter des directives dans le code source C des tâches pour préciser le temps qu'il faudrait pour réaliser la même action en matériel: `srl_busy_cycles` (voir SrlApi).

Il n'existe aucune référence au temps de calcul dans le code C de la tâche `idct` logicielle. En lisant le code de l'implémentation matérielle du coprocesseur `Idct`, déduisez les temps nécessaires aux différentes parties du traitement. our introduire un coprocesseur matériel, il faut commencer par modifier le modèle de la tâche `idct`, en définissant une implémentation matériellel:

 En utilisant un coprocesseur virtuel, pour la tâche `idct`, détermidez quel est le gain en performances apporté par le coprocesseur, pour différents temps de traitement (1 cycle, 8 cycles, 64 cycles, 512 cycles ou 2048 cycles pour traiter un bloc de 64 pixels).quel est le temps nécessaire pour décoder 25 images ?

3. Coprocesseur matériel

Remplacez la déclaration `Synthetic()` par une déclaration de coprocesseur matériel virtuel `HwTask( IdctCoproc )`.

 Quel est maintenant le temps de simulation nécessaire pour 25 images ?

 Qu'en déduisez-vous sur la différence entre les deux possibilités pour tester une implémentation matérielle ?

 Quel intérêt y a-t-il à pouvoir caractériser précisément le temps de traitement d'une tâche matérielle à partir d'un code en C ?

4. Compte-Rendu

Comme pour les TP précédents, vous rendrez une archive contenant:

```
$ tar tzf binome0_binome1.tar.gz
tp5/
tp5/rapport.pdf
tp5/vgmn_noirq_multi.py
tp5/mjpeg/
tp5/mjpeg/mjpeg.py
tp5/mjpeg/src/
tp5/mjpeg/src/iqzz.c
tp5/mjpeg/src/idct.c
tp5/mjpeg/src/libu.c
```

Cette archive devra être livrée avant le mardi 13 mars 2007, 18h00 à [MailAsim:nipo Nicolas Pouillon]