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 inutile 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 l'implantation de cette tâche comme un processeur matériel spécialisé.

Pour éviter de re-écrire toute l'application logicielle, on s'impose que le coprocesseur matériel utilise les mêmes canaux de communication (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'éta 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, on utilise pour cela un composant matériel générique, appelé contrôleur MWMR qui est un initiateur VCI, capable de lire ou d'écrire dans différentes FIFO MWMR, et qui fournit au coprocesseur autant de canaux de communications que celui-ci en a besoin. C'est ce même composant matériel qui est utilisé par les composants RAMDAC et TG.

Nous repartirons de la plateforme du TP3: VgmnNoirqMulti. Nous modifierons cette plateforme comportant deux processeurs Mips, pour ajouter un coprocesseur matériel dédié au traitement de l'`Idct`.

Reprenez les fichiers du TP2:

- La description de la plateforme
- La description de l'application
- Le code des tâches (`Libu` ne gère qu'un seul pipeline et `Split` n'existe pas)

? Pour un déploiement entièrement logiciel, rappelez le temps nécessaire pour décoder 25 images.

1. Modification du TCG

Maintenant nous allons utiliser un coprocesseur dédié au traitement de l'`Idct`.

Ajoutez une déclaration à la tâche `idct`:

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

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

2. Tâche matérielle virtuelle

Il existe plusieurs solutions micro-architecturales pour la synthèse d'un coprocesseur spécialisé. Dans le cas d'une transformation IDCT, suivant le nombre d'opérateurs arithmétiques utilisés, on peut effectuer le calcul d'un bloc de 64 pixels en un cycle, 8 cycles, 64 cycles ou 256 cycles. Le coût matériel est inversement proportionnel au temps de calcul.

Avant de se lancer dans la synthèse, il faut donc évaluer précisément les temps de calcul requis pour le coprocesseur, une fois celui-ci placé dans son environnement de travail. On commence donc par *encapsuler* la tâche matérielle `idct` dans un composant matériel générique appelé *threader*. Ce composant s'interface d'un côté avec le contrôleur MWMR, et de l'autre avec la tâche logicielle

2.1. Déploiement

Ici, la tâche sera virtuellement matérielle dans le sens où nous utiliserons le code C de notre tâche pour implémenter le comportement matériel de l'`Idct`.

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

L'implémentation `Synthetic()` doit être accompagnée d'une déclaration de tâche logicielle (`SwTask`) et permet la synthèse virtuelle de la tâche. On peut alors déployer la tâche comme si elle était matérielle, son comportement est simulé.

 Avec cette tâche déployée en coprocesseur virtuel, quel est le temps nécessaire pour décoder 25 images ? Quel rapport à la simulation avec un vrai coprocesseur matériel ?

2.2. Précision temporelle

Pour rendre 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).

En lisant le code de l'implémentation matérielle du coprocesseur `Idct`, déduisez les temps nécessaires aux différentes parties du traitement.

2.3. Comparaison des implémentations

Ajoutez les directives temporelles dans `src/idct.c`.

 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 ?

3. 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]