TP5: 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 utiliser les mêmes canaux de communication MWMR 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 de l'outil de synthèse de coprocesseur, 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.

Vous repartirez de la plateforme du TP3: VgmnNoirqMulti, pour une architecture comportant 3 processeurs, et vous modifierez cette architecture, 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 processeur 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'un déploiement utilisant 2 processeurs, lorsque la tâche `idct` est placée sur le premier processeur, que la tâche `vld` est placée sur le 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 1 cycle ou en 1000 cycles. En première approximation, le coût matériel est proportionnel au le nombre d'opérateurs arithmétiques travaillant en parallèle, et ce nombre 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 encapsulée.

mettre ici le dessin contenant le threader

Pour utiliser un tel coprocesseur *virtuel*, 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' ] ),
                                    SyntheticTask()   ] )
```

- 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)
```

Après synthèse, 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.

Q2. Combien de coefficients sont transférés par cycle sur l'interface FIFO d'entrée? Combien de pixels sont transférés par cycle sur l'interface FIFO de sortie? En déduire les durées minimales (en nombre de cycles) pour les étapes 1 et 3 ci-dessus.

Les temps de communication correspondant aux étapes 1 et 3 sont précisément décrits par le simulateur SystemC, qui reproduit (cycle par cycle) le comportement des interfaces FIFO entre le threader et le contrôleur MWMR (y compris en cas de contention pour l'accès à la mémoire).

En revanche, le nombre de cycles nécessaires pour exécuter l'étape 2 ci-dessus (temps de calcul "interne" à la tâche logicielle) n'est pas défini par le code de la tâche logicielle. Si on ne précise rien, cela correspond à un temps d'exécution du calcul en "zéro" cycles. Pour préciser un nombre de cycles d'exécution, il faut modifier le code C de la tâche `idct`, et insérer, entre les deux primitives *srl_mwmr_read()* et *srl_mwmr_write()*, un appel à la la fonction bloquante *srl_busy_cycles(ncycles)*. L'argument *ncycles* est le nombre de cycles d'attente entre les deux primitives de communication, et il modélise donc le temps de calcul (voir SrlApi).

```
srl_mwmr_read();
...
srl_busy_cycles( n );
...
srl_mwmr_write();
```

Q3. pour quelle raison peut-on affirmer sans aucune expérimentation (c'est à dire sans aucune simulation), qu'il est sans intérêt de synthétiser un coprocesseur matériel dont le temps de calcul soit inférieur à une centaine de cycles?

Modifier la description DSX pour déployer l'application MJPEG sur une architecture comportant 2 processeurs MIPS et un coprocesseur *virtuel* pour la tâche `idct`.

Q4. Mesurez le nombre de cycle pour décompresser 25 images, en faisant varier la valeur du paramètre *ncycles* de la fonction *srl_busy_cycles()*, dans le code C de la tâche `idct`. On essaiera les valeurs 8, 64, 512, et 4096 cycles. En déduire un objectif de performance "raisonnable" pour la synthèse du coprocesseur IDCT.

3. Coprocesseur matériel

On va maintenant utiliser un "vrai" coprocesseur matériel IDCT, disponible dans la bibliothèque SoCLib. Ce coprocesseur matériel est générique, en ce sens qu'on peut paramètrer le nombre de cycles pour effectuer la transformation d'un bloc de 64 pixels. Les valeurs possibles de ce paramètre sont 8, 64, 512, et 4096 cycles.

Remplacez dans le modèle DSX de la tâche `idct`, la déclaration `SyntheticTask()` par une déclaration de coprocesseur matériel `HwTask( IdctCoproc )`, et relancez la simulation de cette nouvelle plate-forme, pour les 4 valeurs possibles du paramètre.

Q5. Comment expliquez-vous les différences entre les performances obtenues, suivant qu'on utilise un processeur réel ou virtuel?

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]