TP5: Introduction d'un coprocesseur matériel spécialisé

1. 0. Objectif
2. 1. Coprocesseur virtuel
3. 3. Coprocesseur matériel
4. 4. Compte-Rendu
5. Suite

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'Amdahl 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.

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)

? Rappelez le temps nécessaire pour décoder 25 images, dans le cas d'un déploiement utilisant 3 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. 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 - 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 qui est un service fourni par l'environnement DSX.

- du côté matériel, ce composant s'interface avec le composant matériel *contrôleur MWMR*
- du côté logiciel, ce composant dialogue avec un processus hébergeant la tâche logicielle `idct`.

Ceci permet une utilisation du code existant - sans modification - 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*. Le premier processus est le simulateur SystemC modélisant l'architecture matérielle (y compris le contrôleur MWMR et le composant d'émulation), le second processus est la tâche logicielle encapsulée.

Pour utiliser un tel coprocesseur *virtuel*, il faut modifier trois choses dans la description DSX:

- dans la définition du modèle de la tâche `idct`, il faut ajouter l'implémentation `SyntheticTask()`. Le coprocesseur matériel étant paramètrable, il faut également définir un nouveau paramètre `EXEC_TIME` dans la liste des paramètres de la tâche `idct`. Ce paramètre permet de spécifier le nombre de cycles utilisés par le coprocesseur matériel pour effectuer la transformation IDCT d'un bloc de 64 pixels.

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

- La valeur du paramètre EXEC_TIME doit être définie au moment où on instancie la tâche `idct` dans le TCG.

```
Task( 'idct0', idct,
      portmap = { 'output':idct_libu,
                  'input' :iqzz_idct },
```

```
            defines = { 'XSIZE':'48', 'YSIZE':'48', 'EXEC_TIME':'64' }
        )
```

- 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("idct0", vci = mapper.hard.vgmn)
```

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

1. lecture d'un bloc de 64 coefficients du canal MWMR d'entrée vers une mémoire locale,
2. calcul d'un bloc de 64 pixels, et stockage de ces pixels dans la mémoire locale (distinct du bloc d'entrée),
3. écriture de ces 64 pixels de la mémoire locale vers le canal MWMR de sortie.

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 coprocesseur émulé et le contrôleur MWMR (y compris en cas de contention pour l'accès à la mémoire).

? 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.

Le nombre de cycles nécessaires pour exécuter l'étape 2 ci-dessus (temps de calcul pour un bloc de 64 pixels) est défini par la valeur du paramètre EXEC_TIME, mais si on ne modifie pas le code C de la tâche `idct`, ce calcul sera effectué "en temps nul" du point de vue du matériel.

Pour préciser le nombre de cycles d'exécution pour l'étape 2, 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()*.

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

L'argument définit le nombre de cycles que prendrait une version matérielle du même traitement, et modélise donc le temps de calcul (voir SrlApi). EXEC_TIME est issu des paramètres `defines` de la tâche.

? 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`.

? Mesurez le nombre de cycle pour décompresser 25 images, en faisant varier la valeur du paramètre *EXEC_TIME*. On essaiera les valeurs 8, 64, 512, et 4096. 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, pour lequel il existe un modèle de simulation dans la bibliothèque SoCLib. Ce coprocesseur matériel est générique, en ce sens qu'on peut paramètrer le nombre de

cycles pour effectuer la transformation IDCT d'un bloc de 64 pixels. Les valeurs possibles de ce paramètre sont les suivantes:

- 16 cycles : le coprocesseur contient 29 additionneurs et 11 multiplieurs.
- 96 cycles : le coprocesseur contient 5 additionneurs et 3 multiplieurs.
- 240 cycles : le coprocesseur contient 2 additionneurs et 1 multiplieur.
- 464 cycles : le coprocesseur contient 1 additionneur et 1 multiplieur.

Ce paramètre porte le même nom (EXEC_TIME) que pour le coprocesseur virtuel. Ceci permet d'avoir une description homogène.

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.

? Quelles différences de performance observez-vous suivant qu'on utilise un processeur réel ou virtuel ?

? Quel intérêt a-t-on à utiliser un coprocesseur virtuel pendant les phases d'exploration architecturale ?

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]

Suite

TP Suivant: MjpegCourse/Synthese