TP3: Déploiement de l'application MJPEG sur architecture multiprocesseur

TP Précédent: MjpegCourse/Monopro

0. Objectif

Comme pour les précédents TP, vous aurez à fournir un certain nombre de sources et un rapport. Il vous est conseillé de parcourir la documentation dans le trac, en partant de WikiStart. Par ailleurs, vous êtes nombreux à faire des erreurs connues?.

La première partie de ce TP vise la description d'une architecture matérielle multiprocesseurs générique, appelée VgmnNoirqMulti. Cette architecture est générique dans le sens où on peut faire varier par un simple paramètre le nombre de processeurs et le nombre de bancs mémoire, ainsi que les caractéristiques des caches rattachés aux processeurs.

La seconde partie de ce TP vise à déployer l'application MJPEG sur cette plateforme générique en faisant varier les paramètres dont on dispose. Cette *exploration architecturale* est le but réel de l'outil DSX.

1. Description de l'architecture générique VgmnNoirqMulti

L'architecture VgmnNoirqMono utilisée dans le TP2 était une architecture mono-processeur non-paramètrable. On se propose de décrire maintenant avec DSX une architecture multi-processeur générique, dont les paramètres sont :

- Le nombre de processeurs : `nproc`
- Le nombre de bancs mémoire : `nram`
- Le nombre de lignes des caches : `icache_nline` et `dcache_nline`
- Le nombre de mots par ligne des caches : `icache_nword` et `dcache_nword`

Cette architecture comportera également un contrôleur de terminaux (MultiTty).

Vous pourrez vous inspirer de l'architecture VgmnNoirqMono définie dans le TP2, en n'hésitant pas à utiliser les constructions du langage Python permettant d'exprimer la généricité (tableaux, boucles, etc.). Cette architecture générique VgmnNoirqMulti sera décrite dans un fichier DSX séparé, pour faciliter sa réutilisation.

Pour pouvez ajouter des paramètres à la fonction définissant l'architecture. Des valeurs par défaut peuvent être spécifiées.

```
def VgmnNoirqMulti( nproc, nram, icache_nline = 16, icache_nword = 8, dcache_nline = 16, dcache_
```

Vous validerez cette architecture générique VgmnNoirqMulti, en déployant l'application MJPEG sur une instance particulière de cette architecture équivalente à celle utilisée dans le TP2 (c'est à dire un seul processeur et deux bancs mémoire). On utilisera le système d'exploitation embarqué Mutek/S. On utilisera deux bancs mémoire, et des caches de 64 lignes de 8 mots.

On instanciera l'architecture avec la ligne:

```
archi = VgmnNoirqMulti( nproc = 1, nram = 2,
                        icache_nline = 64, icache_nword = 8,
                        dcache_nline = 64, dcache_nword = 8 )
```

 Q1: Combien faut-il de cycles pour décompresser 25 images?

2. Exploration Architecturale

Dans cette seconde partie, on va déployer l'application MJPEG sur l'architecture MPSoC VgmnNoirqMulti. On va principalement faire varier le nombre de processeurs et la répartition des tâches logicielles sur les processeurs. Les deux tâches d'entrée/sortie `tg` et `ramdac` sont toujours implantées sur des coprocesseurs matériels spécialisés, et il ya donc 5 tâches "logicielles" à déployer sur un nombre de processeurs qui reste à déterminer.

2.1 Profilage de l'application

Pour guider la répartition des tâches sur les processeurs, on commence par effectuer un profilage de l'application sur station de travail POSIX, en mesurant les temps passés dans les différentes tâches.

La mise en oeuvre du profilage d'une application multithreadée en utilisant les outils actuellement installés sur le réseau est non triviale. Comme elle ne représente pas un intérêt particulier pour le TP, on vous fournit un script pour réaliser le profilage. Il s'utilise avec deux paramètres: le temps pendant lequel il faut lancer le programme, et le programme à lancer.

```
$ ~pouillon/opt/bin/gprof-mt 15 ./exe.posix
```

Dès qu'il parvient à obtenir les résultats, le script vous rend sur la première colonne le temps relatif (en %) passé dans chacunes des 5 tâches `demux`, `vld`, `iqzz`, `idct` et `libu`.

 Q2: Donnez l'estimation des temps de calcul relatifs pour chacune des 5 tâches "logicielles" de l'application MJPEG. Qu'en déduisez-vous sur les façons optimales de déployer MJPEG sur 2, 3, 4 et 5 processeurs ?

2.2 Déploiement sur une architecture à 5 processeurs

Déployez l'architecture MJPEG sur une architecture VgmnNoirqMultiPro comportant cinq processeurs, (c'est à dire une tâche par processeur) et lancez la simulation.

 Q3: Combien faut-il de cycles pour décompresser 25 images?

On cherche maintenant à estimer le taux d'utilisation de chacun des 5 processeurs.

 Q5: Quel est selon vous le processeur le plus chargé ?

2.3 Déploiement sur des architectures à 4, 3 et 2 processeurs

Déployez l'application MJPEG sur des architectures matérielles comportant 4, puis 3, puis 2 processeurs, en utilisant les informations de profilage et de taux d'utilisation obtenues dans les questions précédentes, pour placer *intelligemment* les tâches de façon à obtenir les temps de décompression les plus courts possibles.

Q6: Quel est le nombre de cycles minimal pour décompresser 25 images avec 4 processeurs ?

Q7: Quel est le nombre de cycles minimal pour décompresser 25 images avec 3 processeurs ?

Q8: Quel est le nombre de cycles minimal pour décompresser 25 images avec 2 processeurs ?

2.4 Influence de la taille des caches

On souhaite maintenant évaluer l'influence de la taille des caches sur le temps de décompression.

Pour faciliter l'exploration architecturale sans avoir à modifier le source de votre description à chaque fois que vous voulez changer un paramètre, il serait judicieux d'utiliser la ligne de commande lors de l'interprétation de la description. Dans ce but, on fournit un fichier attachment:cmdline_parser.py . Une fois placé dans le répertoire de votre description DSX, il s'utilise de la manière suivante:

```
from cmdline_parser import *
```

Il définit les variables suivantes:

- `dcache_lines`, nombre passé sur la ligne de commande après le flag `--dl`, à défaut, 1024.
- `icache_lines`, nombre passé sur la ligne de commande après le flag `--il`, à défaut, 1024.

Utilisez ces variables à l'instanciation de votre plateforme. Pour créer une plateforme avec des caches avec 64 lignes de données, il suffira alors d'exécuter:

```
$ ./mjpeg --dl 64
```

On se place dans l'hypothèse d'une architecture à 2 processseurs, en conservant le placement des tâches optimal défini à la question précédente. On utilisera des lignes de cache de 8 mots, et on se contentera de faire varier le nombre de lignes.

- Mesurez le temps de calcul pour décompresser 2 images, en utilisant deux "gros" caches de 1024 lignes, pour les instructions comme pour les données.
- Refaites cette mesure en diminuant progressivement le nombre de lignes du cache de données (256, puis 64, puis 16 puis 4 puis 1), en conservant une capacité de 1024 lignes pour le cache d'instructions.
- Même question en diminuant progressivement le nombre de lignes du cache d'instructions (256, puis 64, puis 16 puis 4 puis 1), en conservantune capacité de 1024 lignes pour le cache de données.

Q9: Regroupez ces résultats dans deux tableaux de synthèse.

Q10: Que choisiriez-vous comme capacité pour les caches, sachant que la surface de la mémoire embarquée est un facteur important du coût de fabrication (on comparera en particulier la capacité des caches à la capacité des bancs mémoire `ram0` et `ram1`).

3. Compte-Rendu

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

```
$ tar tzf binome0_binome1.tar.gz
tp3/
tp3/rapport.pdf
tp3/vgmn_noirq_multi.py
tp3/mjpeg/
tp3/mjpeg/mjpeg.py
tp3/mjpeg/src/
tp3/mjpeg/src/iqzz.c
tp3/mjpeg/src/libu.c
```

Le fichier `mjpeg.py` sera celui de la partie 2.4, avec la prise en compte des arguments sur la ligne de commande. Les deux sources C sont ceux des deux derniers TP, éventuellement modifiés.

Cette archive devra être livrée avant le mardi 4 mars 2008, 18h00 à [MailAsim:nipo Nicolas Pouillon]

Suite

TP Suivant: MjpegCourse/Multipipe