TP4 : Exécution sur architecture multi-cluster

1. 0. Objectif
2. 1. Parallélisation du TCG
3. 2. Architecture matérielle multi-processeur clusterisée
4. 3. Déploiement et exploration architecturale
5. 4. Compte-Rendu
6. Suite

TP Précédent: MjpegCourse/Multipro

0. Objectif

On cherche dans ce quatrième TP à augmenter encore le débit de la chaîne de décompression, pour permettre - par exemple - de traîter des images de plus grandes dimensions, tout en respectant la fréquence video. Cette augmentation de débit peut être obtenue en augmentant la fréquence d'horloge, mais cette approche a évidemment des limites. On essayera donc plutôt d'augmenter le parallélisme de traitement du flux MJPEG.

Pour augmenter le parallélisme, il ne suffit pas d'augmenter le nombre de processeurs dans l'architecture matérielle, il faut également augmenter le nombre de tâches de l'application logicielle, ce qui impose de modifier la structure du TCG.

- La première partie du TP vise la définition d'un graphe de tâches *multi-pipeline*.
- La seconde partie du TP porte sur la définition d'une architecture matérielle *multi-clusters*.
- La troisième partie du TP analyse l'impact du placement des canaux de communication sur les bancs mémoire dans les architectures `NUMA` (Non Uniform Memory Access).

Commencez par créer un répertoire de travail `tp4`, et recopiez dans ce répertoire les différents fichiers et/ou répertoires sources contenus dans le répertoire `tp3`.

1. Parallélisation du TCG

Le TCG défini dans le TP1 et re-utilisé dans les TP2 et TP3 comportait 7 tâches. Il exploitait un parallélisme de type *macro-pipeline*. Différentes tâches traitent différents blocs de la même image: Toutes les tâches s'exécutent en parallèle, mais sur des blocs différents de l'image. Il est difficile d'augmenter le nombre d'étages de ce macro-pipeline, car les tâches les plus coûteuses en temps de calcul (VLD et IDCT) ne se découpent pas facilement en sous-tâches.

On va donc exploiter un autre type de parallélisme en utilisant deux pipelines de décompression. Chaque pipeline traite une image complête. On introduit une tâche chargée de distribuer aternativement aux deux pipe-line le flux MJPEG. Cette nouvelle tâche `split` se situera entre les tâches `tg` et `demux`. La tâche `libu` doit être modifiée pour récupérer alternativement les images décompressées provenant des deux pipelines, avant de les envoyer vers la tâche `ramdac`.

Modifiez la structure du TCG dans la description DSX de l'application. Vous devez introduire un nouveau modèle de tâche pour la tâche `split`, et modifiier le modèle de la tâche `libu`. Il faut ensuite modifier la topologie du TCG en définissant explicitement toutes les intances de tâches et tous les canaux de communication nécessaires.

Le code de la tâche `split` doit analyser octet par octet le flux MJPEG, pour détecter le marqueur de début d'image

(SOI = 0xffd8), de façon à l'aiguiller vers le bon canal de sortie.

Le pseudo core correspondant à l'algorithme de split est:

```
canal de sortie = le premier

toujours:
   b = lire un octet
   si b == 0xff
      m = lire un octet
      si m == 0xd8
         remplir la sortie courante de 0xff
         envoyer le bloc
         changer de canal de sortie
      ecrire b dans la sortie
      ecrire m dans la sortie
      retourner au debut de la boucle
   ecrire b dans la sortie
```

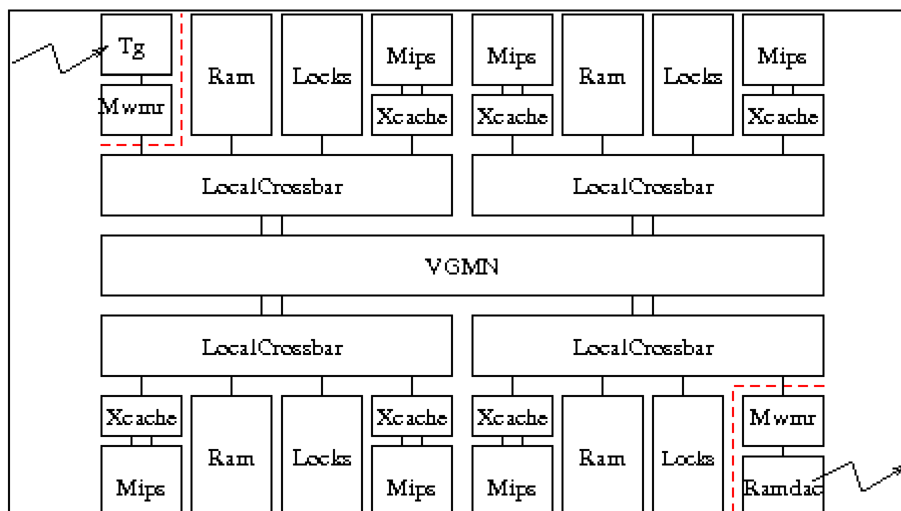
Pour valider fonctionnellement cette nouvelle description de l'application logicielle, déployez-la sur station de travail POSIX. vous devez voir les mêmes images qu'avant, dans le même ordre.

2. Architecture matérielle multi-processeur clusterisée

Pour supporter la charge induite par ces nouvelles tâches, il faut augmenter le nombre d'unités de traitement (processeurs ou coprocesseurs). Pour éviter que l'accès à la mémoire devienne un goulot d'étranglement, il est également souhaitable d'augmenter le nombre de bancs mémoire physique, de façon à répartir les données. Et lorsque le nombre d'entités communicantes (initiateurs ou cibles) augmente, il est utile de structurer l'architecture en sous-systèmes.

Cette structuration a des justifications fonctionnelles:

- On cherche à regrouper dans un même sous-sytème les différents composants matériels qui réalisent une même partie de l'application, et communiquent fortement entre eux.
- Elle facilite également la réalisation matérielle : Chaque sous-système pourra être implanté physiquement dans un même domaine synchrone, et utiliser sa propre horloge, conformément au principe GALS (Globally Asynchronous, Locally Synchronous).



Chaque sous-système constitue un *cluster*, et contient des processeurs, de la mémoire, et dispose de son propre mécanisme d'interconnexion local.

Les différents clusters sont interconnectés entre eux par une micro-réseau à interface VCI/OCB, qui pourra être modélisé par un composant `Vgmn`.

On utilisera comme mécanisme d'interconnexion interne à chaque cluster le composant LocalCrossbar (voir SoclibComponents). Ce composant matériel est un petit crossbar, qui possède un nombre variable de ports *initiateur* et *cible* permettant de connecter les composants matériels appartenant au cluster. Il possède également deux ports *initiateur* (`initiator_to_up`) et *cible* (`target_to_up`) permettant l'accès au micro-réseau.

Cette structuration aboutit donc à l'utilisation d'un mécanisme d'interconnexion à deux niveaux (interconnect global: `Vgmn`, et interconnect local: `LocalCrossbar`), bien que tous les composants matériels (initiateurs et cibles) continuent à partager le même espace d'adressage.

Pour faciliter l'exploration architecturale, on souhaite définir une architecture générique dont les paramètres sont:

- la latence minimale du Vgmn
- le nombre de clusters et, pour chaque cluster,
    - le nombre de bancs mémoire
    - le nombre de processeurs

Utilisez la définition de l'architecture ClusteredNoirqMulti.

3. Déploiement et exploration architecturale

Modifiez la description DSX de l'application MJPEG:

- Remplacez l'instanciation de VgmnNoirqMulti par

```
archi = ClusteredNoirqMulti( cpus = [1, 2, 2, 1],
                             rams = [1, 1, 1, 1],
                             min_latency = 10 )
```

- Ajoutez la création des coprocesseurs `tg` et `ramdac` sur le premier et le dernier cluster

respectivement.

La structure de l'application logicielle (TCG), et l'architecture matérielle étant définies, l'exploration architecturale consiste donc à analyser l'influence du placement des objets logiciels sur les composants matériels. On s'intéresse tout particulièrement au placement des canaux de communication sur les bancs mémoire physiques.

Dans ce type d'architecture multi-clusters, les temps d'accès à la mémoire sont très différents, suivant qu'un processeur adresse la mémoire locale au sous-système, ou à un autre sous-système. On parle d'architecture NUMA (Non Uniform Memory Access).

Refaites le placement des canaux de communication de manière *intelligente*. Essayez ensuite de varier sur le placement du verrou par rapport au placement du canal, de placer le canal plutôt du côté de la consommation, ou de la production, ...

- Redéployez les canaux MWMR et les tâches sur les rams aux noms de la forme `[uc]ram<no cluster>_<no>`

Combien faut-il de cycles pour décompresser 25 images?

? Essayez d'en extraire un critère de performance en fonction des placements.
Pour cette question, si vous trouvez la simulation trop longue pour 25 images, ne vous basez pas sur la simulation pour une image, car le pipeline de traitement est vide au départ. Essayez de prendre au moins 7 images, et d'ignorer le temps de *remplissage* (2 premières images).

4. Compte-Rendu

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

```
$ tar tzf binome0_binome1.tar.gz
tp4/
tp4/rapport.pdf
tp4/mjpeg/
tp4/mjpeg/mjpeg.py
tp4/mjpeg/src/
tp4/mjpeg/src/iqzz/iqzz.c
tp4/mjpeg/src/iqzz/iqzz.task
tp4/mjpeg/src/libu/libu.c
tp4/mjpeg/src/libu/libu.task
tp4/mjpeg/src/split/split.c
tp4/mjpeg/src/split/split.task
```

Cette archive devra être livrée avant le jeudi 29 octobre 2009, 18h00 CEST à [MailAsim:nipo Nicolas Pouillon]

Suite

TP Suivant: MjpegCourse/Coproc