UNIVERSITÉ DE RENNES 1

TP 7-8-9

L'objectif de ce TP est de vous faire utiliser une chaine de développement croisée pour processeur VLIW embarqué. Le processeur utilisé dans ce TP est un modèle très générique de VLIW basé sur les machines Lx (HP) et ST200 (STMicroelectronics).

La chaîne de développement complète est par ailleurs disponible gratuitement sur le site de HP (http://www.hpl.hp.com/downloads/vex/). Un manuel technique vous est également fourni.

Cette chaîne de développement offre les fonctionnalités suivantes

- Un compilateur C optimisant capable de générer un code très efficace pour des variantes de processeurs VLIW (organisés en un nombre configurable de slots).
- Un simulateur de machine qui permet d'obtenir simplement et très rapidement des informations très précises (nombre de cycles, nombre de défauts de cache, etc.) sur l'exécution du programme sur la machine.

Dans ce TP, nous mettrons l'accent sur l'utilisation judicieuse de la hiérarchie mémoire de la machine cible lorsque l'on souhaite mettre en œuvre des algorithmes de traitement d'image temps réel.

1. Description de l'algorithme

Le traitement à implanter sur la machine est une opération de détection de contour des objets présents sur une image. Ce type d'opération est très utilisé comme pré-traitement pour des applications de reconnaissance de forme, de vision par ordinateur, etc.

L'approche utilisée dans ce TP se base sur l'utilisation d'un filtrage de Deriche, qui permet de réaliser un traitement assez complexe tout en minimisant le nombre de calculs à effectuer. La trame générale de l'algorithme est donnée Figure 1.

L'algorithme consiste en une succession de convolutions récursives simples, opérant sur les lignes, les colonnes, et parcourant l'image dans les deux sens (droite/gauche, haut/bas)

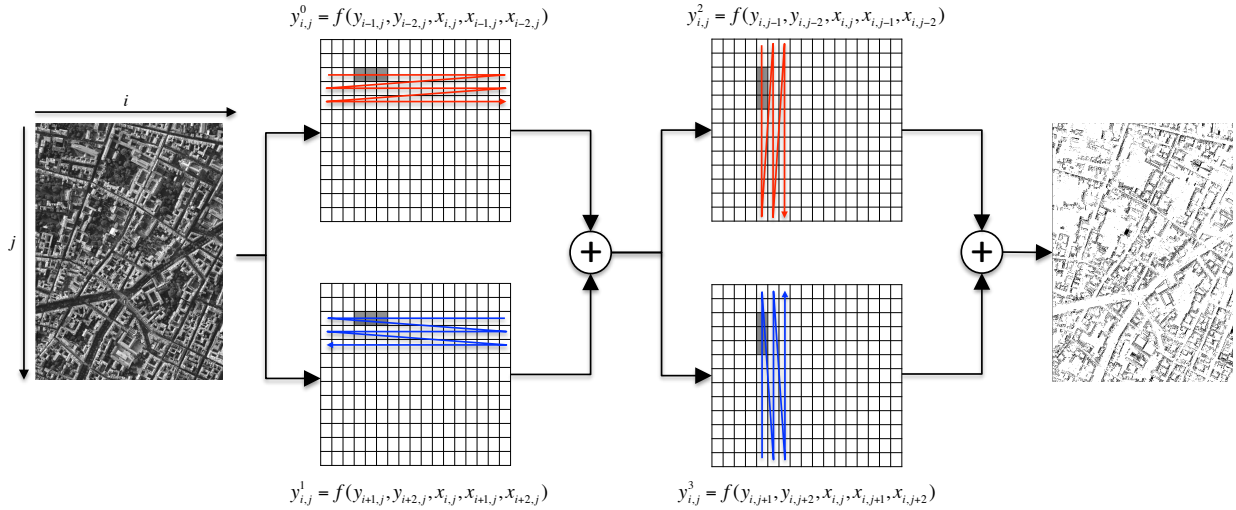
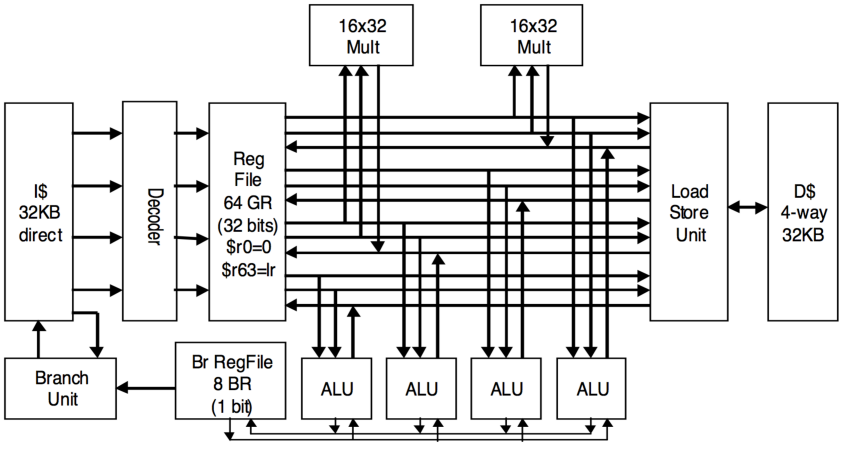


Figure 1 : Algorithme filtrage de Deriche

2. Description de la machine cible.

Le processeur vex utilisé dans ce TP est une machine VLIW à quatre slots d'exécution. Elle permet de réaliser une lecture et une écriture, jusqu'à quatre opérations arithmétiques simples et deux multiplications à chaque cycle. Le vex dispose également de deux mémoires caches séparées (une pour le code, l'autre pour les données) dont les caractéristiques peuvent être modifiés par un fichier de configuration.



(e) schéma du processeur vex (ici avec 4 slots d'exécution)

2. Mise en place de l'environnement de développement

L'environnement de développement du processeur VEX ne fonctionne que sur des machines Linux 32 bits, or les machines linux de l'ISTIC sont toutes des machines 64 bits. Afin de pouvoir utiliser la chaine de développement VEX, vous allez devoir vous connecter à une machine virtuelle (machine **`tpase.istic.univ-rennes1.fr`**) sur laquelle vous réaliserez vos développements et mise au point. Une dizaine de comptes (**`tpase0`**, **`tpase1`**, etc..) y ont été crées à cet effet. Les mots de passe vous seront donnés pendant la séance de TP.

2.1. Connexion à la machine virtuelle

Pour accéder à la machine et bénéficier d'une interface graphique, il vous faudra saisir la commande :

```
ssh –X tpase0@tpase.istic.univ-rennes1.fr
```

2.1. Copie de données de/vers la machine virtuelle

Une fois connecté, vous pourrez copier le répertoire **`tp-vex`** depuis votre compte ISTIC vers la machine distante à l'aide de la commande

```
scp –r t "tp-vex " tpase0@tpase.istic.univ-rennes1.fr:
```

Vous pourrez faire de même depuis la machine virtuelle en copiant le répertoire **`tp-vex`** vers votre compte ISTIC à l'aide de la commande

```
scp –r t "tp-vex " login@cassius.istic.univ-rennes1.fr:
```

Attention à ne pas vous tromper ! Vous risquez d'effacer vos modifications par mégarde !

3. Mise en œuvre logicielle naïve

Une première version de l'application décrite en C vous est fournie dans le répertoire **`src/`**. Le fichier le plus important est **`deriche.c`** qui contient la fonction **`deriche_float(…)`**.

Cette version permet de lire une image (au format **`pgm`**) d'y appliquer une détection de contours et de stocker le résultat dans un autre fichier (également au format **`pgm`**). Dans sa version actuelle l'algorithme est mis en œuvre à l'aide d'arithmétique flottante, et correspond à une mise en œuvre naïve de l'algorithme au complet.

3.1. Exécution (et mise au point) sur le processeur hôte.

Pour exécuter l'application sur la machine x86, il suffit de se placer dans le répertoire **`x86/`** et de lancer la commande **`make float`**. Un fichier **`satellite_deriche.pgm`** contenant l'image des contours est alors créé dans le répertoire ou a été lancé la commande.

3.2. Exécution (et mise au point) sur le processeur cible.

Pour exécuter l'application sur la machine VLIW , il suffit de se placer dans le répertoire **`vex/`** et de lancer la commande **`make float`**. Un fichier **`satellite_deriche.pgm`** contenant l'image des contours est alors créé dans le répertoire ou a été lancé la commande.

Pour la suite du TP, il vous est fortement conseillé de réaliser d'abord sur la machine hôte les transformations algorithmiques, puis ensuite de vérifier leur bon fonctionnement avant d'en évaluer l'impact sur les performances de la machine cible.

La commande **`make float`** appele le **`makefile`** qui va compiler le programme pour le vex, produire un fichier assembleur VLIW (**`deriche.s`**) puis lancer le simulateur de processeur VLIW pour exécuter le code machine obtenu.

A l'issu de l'exécution vous pourrez vérifier que les résultats de l'exécution sur le x86 et sur le VLIW donnent le même résultat (fichier **`satellite_deriche.pgm`** produit dans le répertoire **`/vex`**).

De plus, à l'issu de la simulation, le simulateur produit un fichier (ici **`tp_ase.log.float`**) qui permet d'analyser (d'un point de vue quantitatif) l'efficacité de l'implémentation à l'aide de nombreuses métriques (nombre de cycles, durée d'exécution simulée, taux d'utilisation des slots d'exécution, défauts de cache, etc.).

Questions

1. Donnez le nombre de cycles utilisés par l'exécution du programme, et retrouvez sa durée d'exécution. Déduisez en la fréquence de fonctionnement du processeur.
2. Quel est le taux moyen d'utilisation des slots du processeur VLIW, pourquoi le rapport propose-t-il deux chiffres, d'ou eut venir la différence entre ces chiffres ?
3. Combien d'accès mémoire ce programme a-t-il réalisé ?
4. Combien de défauts de cache (pour les données) le programme a-t-il engendré ? Comment pouvez vous expliquer ce chiffre ? La moitié de ces défauts est engendré par des *write-back*, expliquez ce résultat.
5. Combien de défauts de cache (pour les instructions) le programme a-t-il engendré ? Comment pouvez vous expliquer ce chiffre ?
6. La métrique IPC signifie Instruction Per Clock cycle, Que pouvez vous dire quand à l'efficacité de cette implémentation ?
1. Une fois la transformation validée, comparez les rapports d'exécution des deux versions, que pouvez vous observer ? Comment expliquez vous ces résultats ?

Travail à réaliser

1. Transformation flottant vers fixe

1. Complétez le code la fonction **`deriche_slow(…)`**, en la reprogrammant de manière à ne plus utiliser d'arithmétique flottante tout en permettant de conserver la fonctionnalité de l'algorithme. Comme l'indique le choix de codage des coefficients, le format virgule fixe qui vous est suggéré est $Q_{23,8}$.

2. Amélioration de la localité temporelle des accès mémoire pour le cache

2. En supposant qu'un ligne de cache contient 16 octets, combien de lignes de caches doivent être chargées en mémoire entre l'écriture dans **`qy1[i][j]`** au sein de la première boucle (label L1) et sa lecture dans la boucle associée au label L3 ? Même question pour **`qy2[i][j]`** ?
3. Sachant que la taille du cache utilisé est donnée dans le fichier ./vex/vex.cfg, que pouvez vous en déduire sur les chances d'observer un défaut de cache lors de la relecture de **`qy1[i][j]`** en L3 ?
4. Quelle transformation de boucle peut-être utilisée pour rapprocher l'utilisation de la valeur de **`qy1[i][j]`** de son initialisation/écriture ?
5. Mettez en œuvre cette transformation dans la fonction **`deriche_fast()`** du fichier **`deriche.c`**, en validant son fonctionnement (commande **`make fast`**) sur x86 puis vex.