UNIVERSITÉ DE RENNES 1

TP 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/).

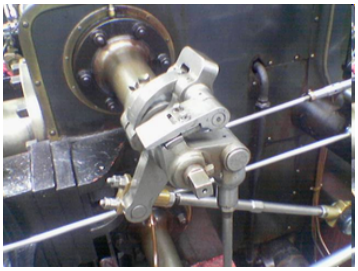
Cette chaîne de développement offre les fonctionnalités suivantes

- 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.

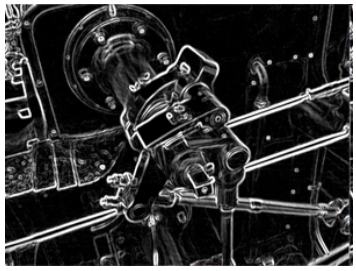
Dans ce TP, nous mettrons l'accent sur l'utilisation judicieuse de la hiérarchie mémoire de la machine cible lorsque l'on souhaite mettre en oeuvre des algorithmes de traitement d'image temps réel.

1. Description de l'application

Le traitement à implanter sur la machine est une opération 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 stéréovision, etc. L'approche utilisée dans ce TP se base sur l'utilisation de filtres de Sobel, qui permettent d'évaluer le gradient vertical et horizontal de l'image.

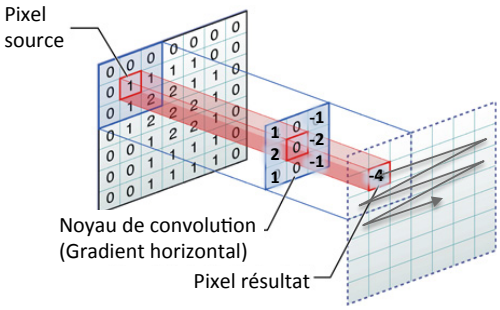


(a) image originale

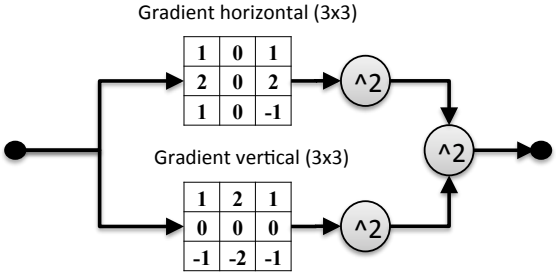


(b) après détection de contours

Le calcul d'un gradient horizontal et/ou vertical est mis en œuvre à l'aide d'un noyau de convolution à deux dimensions, dont le principe de fonctionnement est illustré ci-dessous.



(c) convolution d'image

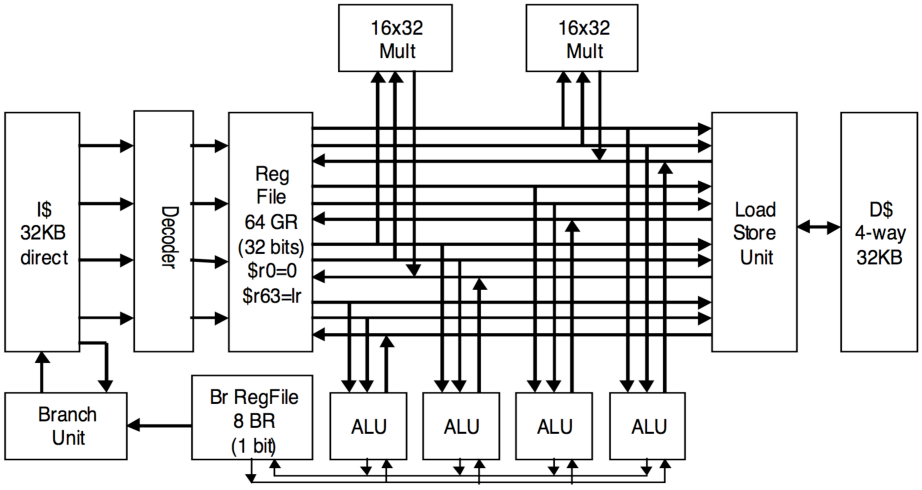


(d) schéma de l'algorithme

Une fois les valeurs de gradient horizontal G_x et vertical obtenues G_y, on calcul la norme L2 du vecteur obtenu. Cette norme permet d'obtenir une estimation fidèle de l'intensité des contours de l'image. Les étapes de cet algorithme sont représentées sur la figure d.

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 opération 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 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 **`edge_detect.c`** qui contient la fonction **`sobel_slow(…)`**.

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. 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 run`**. Un fichier **`sobel.pgm`** contenant l'image des contours est alors créé dans le répertoire ou a été lancé la commande.

Travail à effectuer :

1. Transformation flottant vers fixe

1. Modifiez le programme de manière à ne plus utiliser d'arithmétique flottante tout en permettant de conserver la fonctionnalité de l'algorithme (vous pourrez observer que le programme ne manipule presque que des valeurs entières).

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

1. En supposant qu'un ligne de cache contient 8 octets, combien de lignes de caches doivent être chargées en mémoire entre l'écriture dans **`Gx[i][j]`** au sein de la première boucle (associé au label L1) et sa lecture dans la boucle associée au label L2.
2. Que pouvez vous en déduire sur les chances d'observer un défaut de cache lors de la relecture de **`Gx[i][j]`** en L2 ?
3. Quelle transformation de boucle peut-être utilisée pour rapprocher l'utilisation de la valeur de **`Gx[i][j]`** de son initialisation/écriture ?
4. Mettez en œuvre cette transformation dans la fonction **`sobel_fast()`** du fichier **`edge_detect.c`**, en validant son fonctionnement (commande **`make fast`**) sur x86 puis vex.