# Hiérarchie mémoire et transformation de code

S. Mancini

Les réponses aux questions sont à insérer dans le fichier `rapport/rapport.tex` de l'archive distribuée.

## 1 Rappels et questions de cours

▷ **Question 1 :**
Quelles sont les hypothèses sur lesquelles se basent les mécanismes de mémoire cache ?

▷ **Question 2 :**
Quel est l'avantage/inconvénient des caches associatif par rapport aux caches à correspondance directe ?

▷ **Question 3 :**
Donner les trois types de défaut de cache qui peuvent se produire dans un cache associatif.

Un processeur PPC 440 dispose d'un cache dont les caractéristiques sont les suivantes :

| | |
|---|---|
| Taille | 32 KOctet |
| Ligne | 64 Octet |
| Associatif par groupe | 64 voies |

▷ **Question 4 :**

— Quel est le nombre de lignes de cache ?
— Quel est le nombre de groupes associatifs ?

▷ **Question 5 :**

— Quels sont les bits d'adresse utilisés pour calculer le numéro du groupe ?
— Quelle est l'écart minimum des adresses de deux données situées dans deux lignes différentes et dans le même groupe ?

## 2 Performance d'une hiérarchie mémoire

### 2.1 Contexte de l'étude

On s'intéresse à la performance des accès mémoire du programme `rotation` dont le code est donné à la fin du sujet, page 7. La performance d'une hiérarchie mémoire est mesurée en fournissant la séquence d'accès à la mémoire à un simulateur de cache dénommé `Dinero`. Un descriptif du logiciel `Dinero` est consultable sur la page `http://pages.cs.wisc.edu/~markhill/DineroIV/`.

Le programme `rotation` génère une liste de coordonnées $(x, y)$ écrite dans un fichier, appelé *liste d'index* dans la suite. Cette liste d'index est transformée en une liste d'adresses qui sera fournie à `Dinero`. `Dinero` retourne les mesures suivantes :

— Le taux de défaut de cache
— La quantité de données chargées depuis la mémoire principale

▷ **Question 6 :**
En deux phrases, expliquer pourquoi le taux de défaut de cache à lui seul ne permet pas de comparer les performances de deux caches pour une même séquence d'accès ?

Le programme `rotation` est en fait une transformation du programme `rotation_original`.

▷ **Question 7 :**
Expliquer pourquoi le programme `rotation_original` effectue une rotation d'image 2D ?

▷ **Question 8 :**
Vérifiez que `rotation` et `rotation_original` sont bien équivalents. `lx` et `ly` correspondent à la taille de l'image destination (et source aussi). A quoi correspondent `tx` et `ty` ?

Dans la suite, on fixe `lx` et `ly` à 512 et, de plus, `tx` et `ty` seront toujours égaux.

Les images sont stockées en mémoire dans l'ordre canonique, c'est à dire que l'adresse d'un pixel $(x, y)$ est :

$$adr = x + \texttt{lx} * y \tag{1}$$

Lorsque les axes sont centrés sur l'image, on utilise plutôt l'équation

$$adr = x + \frac{\texttt{lx}}{2} + \texttt{lx} * (y + \frac{\texttt{ly}}{2}) \tag{2}$$

# 3 Installation

## 3.1 Installation du simulateur de cache

Créez un répertoire dans lequel vous décompresserez l'archive `TP_Cache.tar.gz`. Dans le répertoire `TP_Cache`, décompresser à nouveau l'archive du simulateur de cache `Dinero` (`d4-7.tar.gz`). Une fois cette dernière archive décompressée, le répertoire `d4-7` contient une page de manuel lisible avec la commande :

```
> man ./d4.1
```

Afin de compiler le simulateur `dineroIV`, depuis le répertoire `d4-7`, faire :

```
> ./configure
> make
```

Afin de tester `dineroIV`, depuis le répertoire `d4-7` lancer :

```
> ./dineroIV -informat d -l1-dbsize 64  -l1-dsize 16k -l1-dassoc 2 << _fin
> 0 0x0
> 0 0x1
> 0 0x40
> 0 0x0
> _fin
```

Le format d'entrée est une ligne par accès mémoire de la forme

```
0 adresse
```

le premier `0` indiquant que l'accès est une donnée, `adresse` est une adresse en *héxadécimal*.

▷ **Question 9 :**
A quoi correspondent les arguments de la ligne de commande ci-dessus ?

▷ **Question 10 :**
Lire le rapport produit par `Dinero` et vérifier que le taux de miss ainsi que le nombre de données chargées sont corrects.

### 3.2 Installation du code testé

Dans le répertoire `TP_Cache`, compiler le code `rotation.c`, par la commande :

```
> make LDLIBS=-lm rotation
```

Vous obtenez un binaire qui produit une liste d'index. Afin de pouvoir mesurer l'efficacité du cache, il faut transformer cette liste d'index en une liste d'adresses dans le format de `Dinero`. A cette fin le script bash `xy_to_adr` vous est fourni. Ce script calcule les adresses à partir des index selon l'équation 1.

## 4 Mesure de la localité spatiale

Afin d'estimer à la louche l'intérêt d'un cache, nous allons mesurer le taux de réutilisation des données pour deux angles de rotation. Lancer les commandes :

```
> ./rotation 512 512 1 1 0 > res/idx_1
> ./rotation 512 512 1 1 0.52 > res/idx_2
```

▷ **Question 11 :**
A l'aide des commandes unix `wc -l` et `sort -u`, trouvez une méthode très simple pour mesurer le taux de réutilisation de chacune des listes d'index.

▷ **Question 12 :**
Expliquez et vérifiez pourquoi le taux de réutilisation ne change pas lorsque la taille de tuile change (faire un essai avec `tx=ty=16`).

## 5 Mesure de l'effet de la taille des tuiles

Le fichier `script_lib` contient un ensemble de fonctions bash qui permettent d'automatiser la mesure de performances. Ces scripts utilisent un répertoire `log` à créer si besoin. Les résultats et expériences peuvent être dans un répertoire `res`. La fonction `boucle` lance des simulations pour différentes valeurs de taille de tuile `t`. Voir le fichier `script_lib` pour le détail des arguments.

### 5.1 Effet de l'angle de rotation sur les performances

Le fichier `rapport.tex` sert de canevas et doit contenir les figures des expériences de la suite du TP. Ouvrir le fichier `script_simple`, analyser son contenu puis le lancer.

▷ **Question 13 :**
Expliquer la forme des deux courbes du fichier `res/expe_simple.pdf`. Si le taux de réutilisation ne change pas avec `t`, expliquez pourquoi le taux de défaut de cache change.

Pour information, le premier argument de la fonction boucle est le nom du fichier des mesures de taux de miss et quantité de données chargées. Ce fichier contient une ligne par valeur de `t` et, dans le cas d'un cache L1, est de la forme :

```
t taux_de_défaut quantité_de_donnee_chargées
```

et, dans le cas d'un cache L2, de la forme :

```
t taux_de_défaut_L1 taux_de_défaut_L2 quantité_de_donnee_chargées_dans_L1 quantité_de_donnee
```

Reproduire l'expérience précédente pour les valeurs de `alpha` suivantes : $(0.52; 1.04; 1.57)$

▷ **Question 14 :**
Expliquer les formes des courbes obtenues.

### 5.2 Influence des paramètres du cache

Reproduire les expériences précédentes pour un cache dont les caractéristiques sont les suivantes :

| | |
|---|---|
| Taille | 16 KOctet |
| Ligne | 32 Octet |
| Associatif par groupe | 2 voies |

▷ **Question 15 :**
Ajoutez les nouvelles figures dans le rapport et commentez les variations de performance observées. Quel est le meilleur cache ?

### 5.3 Impact du schéma d'adressage

Maintenant, on modifie l'ordre de stockage des données en mémoire. La formule de calcul d'adresse est la suivante :

$$adr = (x'\%(\frac{\texttt{lc}}{2})) + (\frac{\texttt{lc}}{2}).(2.(\frac{x'}{(\texttt{lc}/2)}) + (y'\%2)) + 2.\texttt{lx}.(\frac{y'}{2}) \quad (3)$$

Avec
- $x' = x + \frac{lx}{2}$ et $y' = y + \frac{ly}{2}$ lorsque l'origine du repère est au centre de l'image
- $lc$ la longueur de la ligne de cache
- $\frac{a}{b}$ la division entière. Attention $2.\frac{x}{2}$ peut être différent de $x$ !!
- $\%$ est l'opération modulo

Lorsque les axes sont centrés sur l'image, on remplace $x$ et $y$ par $x' = x + \frac{\texttt{lx}}{2}$ et $y' = y + \frac{\texttt{ly}}{2}$.

▷ **Question 16 :**
Pour une ligne de cache de 64 octets, représenter sur un schéma simple la façon dont sont stockés en mémoire les pixels d'une fenêtre $[0, 64] \times [0, 1]$ d'une image $512 \times 512$. Vérifier qu'une ligne de cache contient bien une fenêtre de taille $32 \times 2$.

Complétez le script `xy_to_adr_OBL` afin d'implémenter le schéma d'adressage de l'équation 3. Faire quelques tests sur un fichier qui contienr quelques indices, puis relancer toutes les expériences précédentes, avec `OBL` dans le nom de l'expérience (ce qui permet de lancer ce script automatiquement). Insérer les nouvelles courbes dans le rapport.

▷ **Question 17 :**
Pourquoi les performances sont elles meilleures ?

▷ **Question 18 :**
Comment calculer concrètement ce type d'adresse directement à partir des bits des index $(x, y)$ ?

## 6 Transformation de boucle

En remettant l'application dans son contexte, on s'aperçoit que `alpha` est en fait très petit en pratique (par exemple pour la correction angulaire d'un système de vision).

▷ **Question 19 :**
Comment modifier le programme pour obtenir de meilleures performances ?

▷ **Question 20 :**
Faire la modification et relancer les expériences qui montrent l'amélioration pour l'angle 0.52, pour chacun des caches et schémas d'adressage. Donnez les nouvelles figures et commentez.

# 7 Hiérarchie mémoire

▷ **Question 21 :**
Pour un des caches L1 ci-dessus, mesurer les variations de performance lorsqu'on utilise un cache L2 de 1 MOctets dont la taille d'un bloc est de deux lignes de cache L1 et d'associativié doublé.

Afin de comparer les performances temporelles, on considère un bus de donnée de 32 bits et que la latence pour qu'un cache (L1 ou L2) fasse un accès à une ligne de cache en mémoire externe est de 250 cycles d'horloges. On fait l'hypothèse qu'un accès au cache L2 depuis le cache L1 a une latence de 10 cycles d'horloges.

▷ **Question 22 :**
En utilisant d'une part les taux de miss dans le fichier de résultats produit et les caractéristiques précédentes, comparer les performances temporelles avec ou sans L2.

nov. 22, 13 14:31 **rotation.c** Page 1/1

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

unsigned char*read_img(char nm[], int lx, int ly)
{
  char *img_in, tmp[512];
  FILE *img_in_f;
  int lxi, lyi;
  img_in=malloc(lx*ly);
  img_in_f=fopen(nm, "r");
  fscanf(img_in_f, "%s\n",tmp); // format
  do  // commentaires + taille
        {
          fgets(tmp, 512, img_in_f);
        } while (tmp[0]=='#');
  sscanf(tmp, "%d %d", &lxi, &lyi);
  if ((lx!=lxi) || (ly!=lyi))
        {
          printf("Erreur : lx et ly ne correspondent pas Ã  la taille de l'image %s\n", tmp);
          exit(0);

        }
  do
        {
          fread(tmp,1,1,img_in_f); // dÃ©tection premier 0x0A (fscanf le dÃ©passe de temps en temps)
        } while (tmp[0] != '\n');
  fread(img_in, 1, lx*ly, img_in_f);
  fclose(img_in_f);
  return img_in;
}

int write_img(char nm[], unsigned char *img, int lx, int ly)
{
  FILE *img_out_f;
  img_out_f=fopen(nm, "w");
  fprintf(img_out_f, "P5\n");
  fprintf(img_out_f, "# rotation\n");
  fprintf(img_out_f, "%d %d\n", lx, ly);
  fprintf(img_out_f, "255\n");
  fwrite(img, 1, lx*ly, img_out_f);
  fclose(img_out_f);

  return 0;
}

int main( int argc, char **argv )
{
  float alpha,  m[2][2];
  int lx, ly, tx, ty, x, y,  i,j, ti, tj;
  char *img_in, *img_out;

  /* Initialisation et lecture des arguments */
  if (argc !=6)
        {
          fprintf(stderr, " appel : rotation lx  ly  tx  ty alpha\n");
          exit(1);}
  lx=atoi(argv[1]); ly=atoi(argv[2]);
  tx=atoi(argv[3]); ty=atoi(argv[4]);
  alpha=(float)atof(argv[5]);
  img_in=read_img("input.pgm", lx, ly);
  img_out=malloc(lx*ly);

  m[0][0]=cos(alpha);
  m[0][1]=-sin(alpha);
  m[1][0]=sin(alpha);
  m[1][1]=cos(alpha);

  /* Boucle de calcul */
  for(i=-lx/2; i< lx/2; i+= tx)
        for(j=-ly/2; j< ly/2; j+= ty)
          for(ti=0; ti < tx && i+ti<lx/2;  ti++)
                for(tj=0; tj < ty && j+tj<ly/2;  tj++)
                  {
                        x=m[0][0]*(i+ti)+m[0][1]*(j+tj);
                        y=m[1][0]*(i+ti)+m[1][1]*(j+tj);
                        if (x>-lx/2 && x<lx/2 &&
                              y>-ly/2 && y<ly/2)
                          {
                              img_out[lx/2+i+ti+lx*(ly/2+j+tj)]=img_in[lx/2+x+lx*(ly/2+y)];
                              printf("%d %d\n", x, y);
                          }
                  }
  write_img("output.pgm", img_out, lx, ly);
  return 0;
}
```

Printed by Stephane Mancini

nov. 22, 13 14:25 **rotation_original.c** Page 1/1

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
unsigned char*read_img(char nm[], int lx, int ly)
{
  char *img_in, tmp[512];
  FILE *img_in_f;
  int lxi, lyi;
  img_in=malloc(lx*ly);
#ifndef NO_FILE
  img_in_f=fopen(nm, "r");
  fscanf(img_in_f, "%s\n",tmp); // format
  do  // commentaires + taille
        {
          fgets(tmp, 512, img_in_f);
        } while (tmp[0]=='#');
  sscanf(tmp, "%d %d", &lxi, &lyi);
  if ((lx!=lxi) || (ly!=lyi))
        {
          printf("Erreur : lx et ly ne correspondent pas Ã  la taille de l'image %s\n", tmp);
          exit(0);
        }
  do
        {
          fread(tmp,1,1,img_in_f); // dÃ©tection premier 0x0A (fscanf le dÃ©passe de temps en temps)
        } while (tmp[0] != '\n');
  fread(img_in, 1, lx*ly, img_in_f);
  fclose(img_in_f);
#endif
#ifdef NO_FILE
  int i;
  for(i=0;i<lx*ly;i++) img_in[i]=(i)%255;
#endif
  return img_in;
}

int write_img(char nm[], unsigned char *img, int lx, int ly)
{
  FILE *img_out_f;
#ifndef NO_FILE
  img_out_f=fopen(nm, "w");
  fprintf(img_out_f, "P5\n");
  fprintf(img_out_f, "# rotation\n");
  fprintf(img_out_f, "%d %d\n", lx, ly);
  fprintf(img_out_f, "255\n");
  fwrite(img, 1, lx*ly, img_out_f);
  fclose(img_out_f);
#endif
  return 0;
}


int main( int argc, char **argv )
{
  float alpha,  m[2][2];
  int lx, ly, x, y, i, j;
  char *img_in, *img_out;
  /* Initialisation et lecture des arguments */
  if (argc !=4)
        {
          fprintf(stderr, " appel : rotation lx  ly   alpha\n");
          exit(1);
        }
  lx=atoi(argv[1]);ly=atoi(argv[2]);
  alpha=(float)atof(argv[3]);
  img_in=read_img("input.pgm", lx, ly);
  img_out=malloc(lx*ly);

  m[0][0]=cos(alpha);
  m[0][1]=-sin(alpha);
  m[1][0]=sin(alpha);
  m[1][1]=cos(alpha);

  /* Boucle de calcul */
  for(i=-lx/2; i< lx/2; i++)
        for(j=-ly/2; j< ly/2; j++)
          {
                x=m[0][0]*(i)+m[0][1]*(j);
                y=m[1][0]*(i)+m[1][1]*(j);
                if (x>-lx/2 && x<lx/2 &&
                        y>-ly/2 && y<ly/2)
                  {
                        img_out[lx/2+i+lx*(ly/2+j)]=img_in[lx/2+x+lx*(ly/2+y)];
                        printf("%d %d\n", x, y);
                  }
          }
  write_img("output_orig.pgm", img_out, lx, ly);
  return 0;
}
```

vendredi novembre 22, 2013 1/1