TD-TP1 Méthodes Numériques
Polytech Grenoble, département INFO

Jean-François Méhaut

22 janvier 2020

Le TP1 de MN est à rendre avant le Vendredi 14/2/2020. Vous deposerez sur le moodle une archive contenant le code source et un rapport de 4 pages décrivant le travail que vous avez effectué.

Vous rendrez une archive (.tar.gz) avec les fichiers abr.c, avl.c, des jeux de données (fichiers avec des clés) ainsi qu'un texte décrivant le principe des algos que vous avez implémentés.

1 Préparation du TP

Avant de commencer le TP, vous allez dans un premier temps récupérer les informations sur votre machine.

Le fichier `/proc/cpuinfo` contient des informations sur votre processeur.

Vous pouvez retrouver sur le site Intel les informations détaillées sur votre processeur. Prix du processeur, consommation énergétique, date de mise sur le marché. Vous reporterez ces informations sur le rapport de TP.

Vous pouvez observer la hiérarchie de cache du processeur avec la commande `lstopo`.Si la commande n'est pas trouvée, vous installerez le paquet hwloc (apt install hwloc).

Vous allez récupérer l'archive `POLY.tar.gz` sur le moodle.

Vous allez ensuite exécuter les commandes suivantes `tar` et `make` pour compiler le code source qui vous est fourni..

```
$ tar xvfz poly.tar.gz
poly/
poly/perf_poly.c
poly/p2
poly/poly.h
poly/poly.c
poly/p1
poly/makefile
poly/test_poly.c
$
$ cd poly
```

```
$ make
gcc -Wall -c test_poly.c
gcc -Wall -c poly.c
gcc -o test_poly test_poly.o poly.o
gcc -Wall -c perf_poly.c
gcc -o perf_poly perf_poly.o poly.o
$ test_poly p1 p2
2.000000 + 0.000000 x + 3.000000 X^2 + 4.000000 X^3
0.000000 + 0.000000 x + 1.000000 X^2 + 2.000000 X^3 + 3.000000 X^4
+ 4.000000 X^5 + 5.000000 X^6
$ perf_poly p1 p2
p1 = 2.000000 + 0.000000 x + 3.000000 X^2 + 4.000000 X^3
p2 = 0.000000 + 0.000000 x + 1.000000 X^2 + 2.000000 X^3
+ 3.000000 X^4 + 4.000000 X^5 + 5.000000 X^6
p3 = 2.000000 + 0.000000 x + 4.000000 X^2 + 6.000000 X^3
+ 3.000000 X^4 + 4.000000 X^5 + 5.000000 X^6
addition 3364 cycles
p1+p2 4 operations 0.003092 GFLOP/s
addition 36656 cycles
p4+p5 1025 operations 0.072703 GFLOP/s
$
```

2 Polynômes pleins

Vous allez utiliser la structure ABR présentée en cours et définie dans le fichier `poly.h`. Voici un extrait de ce fichier.

```
typedef struct
 {
  int degre ;
  float *coeff;
 } polyf_t, *p_polyf_t;

p_polyf_t creer_polynome (int degre) ;

void init_polynome (p_polyf_t p, float x) ;

...
...
```

La fonction `creer_polygone` va donc créer un nouveau polynôme. Un tableau de coefficient est alloué. La fonction `init_polynome` permet d'affecter la valeur x à l'ensemble des coefficients du polynome.

```
p_polyf_t creer_polynome (int degre) ;
```

1. Ecrivez la fonction `creer_polynome` qui va retourer un pointeur sur un polynome. Cette fonction va allouer un tabeau pour stocker les coefficients du polynome.

```
void init_polynome (p_polyf_t p, float x) ;
```

2. Ecrivez la fonction `init_polynome` qui affecte la valeur x à tous les coefficients du polynome.

```
void detruire_polynome (p_polyf_t p) ;
```

3. Ecrivez la fonction `detruire_polynome` qui détruit le polynome et desalloue le tableau des coefficients..

```
int egalite_polynome (p_polyf_t p1, p_polyf_t p2) ;
```

4. Ecrivez la fonction `egalite_polynome` qui indique si deux polynomes sont égaux ou pas ? Deux polynomes sont égaux si tous leurs coefficients sont égaux.

```
p_polyf_t addition_polynome (p_polyf_t p1, p_polyf_t p2) ;
```

5. Ecrivez la fonction `addition_polynome` qui calcule la somme de deux polynomes et qui retourne un polynome contenant cette somme.

```
  p_polyf_t multiplication_polynome_scalaire (p_polyf_t p,
                                              float alpha) ;
```

6. Ecrivez la fonction `multiplication_polynome_scalaire` qui calcule un nouveau polynome qui est la multiplication des coefficients du polynome p par la valeur scalaire alpha.

```
float eval_polynome (p_polyf_t p, float x) ;
```

7. Ecrivez la fonction `eval_polynome` qui calcule la valeur du polynome avec la valeur x fournie en paramètre..

```
p_polyf_t multiplication_polynomes (p_polyf_t p1, p_polyf_t p2) ;
```

8. Ecrivez la fonction `multiplication_polynome` qui calcule un nouveau polynome qui est la multiplication des deux polynomes p1 et p2.

```
p_polyf_t puissance_polynome (p_polyf_t p, int n) ;
```

9. Ecrivez la fonction `puissance_polynome` qui calcule un nouveau polynome qui est le polynome p à la puissance n.

```
p_polyf_t composition_polynome (p_polyf_t p, p_polyf_t q) ;
```

10. Ecrivez la fonction `composition _polynome` qui calcule un nouveau polynome qui est la composition de p par q $(p \circ q)$.

3 Polynomes creux

Pour la dernière partie, nous allons chercher à représenter des polynomes creux. Un polynome creux est un polynome où de nombreux coefficients sont nuls. Nous allons chercher à représenter ces polynomes creux en évitnt de stocker les coefficients nuls.

Par exemple, le polynome suivant est de degré 254, mais il n'y a que deux coefficients non nuls.

$p(x) = x^{254} + 2$

Vous devez donc dans un premier temps redéfinir une structure polyome où vous éviterez de stocker des coefficients nuls.

```
typedef struct
 {


 } polyf_creux_t, *p_polyf_creux_t;

polyf_creux_t creer_polynome (int degre) ;

...
...
```

11. Définissez le nouveau type `polyf_creux_t`.

12. Vous choisirez quelques unes des fonctions qui ont été définies sur les polynomes pleins et vous les implémenterez sur des polynomes creux.