Architecture des ordinateurs : TP1

Université de Tours

`Département informatique de Blois`

Travaux pratiques

*
* *

Ce TP est à réaliser en binôme. Il vous est demandé de dresser un rapport contenant votre code et les réponses aux questions lors du rendu de votre travail.

Les différents algorithmes seront écrits en Java ou C.

Arithmétique des ordinateurs

On donne les commandes Java utiles pour la manipulation de données binaires :

```
int x = 0b01101010 // Déclaration de l'entier binaire 01101010 = 106
    x = 0x6a // Déclaration de l'entier hexadécimal 6 * 16 + 10 = 106
int y = 42
Integer.toBinaryString(y) // Imprime la chaine de caractères binaires représentant
                          // y en complément à 2
Float z = 0.1
Float.floatToRawIntBits(z) // Imprime l'entier représentant z sous format IEEE 754
```

Dans la suite de cette partie nous revenons sur les différents éléments du TD1.

Problème de la suite u_n

1. Ré-implémenter l'algorithme du calcul de la suite u_n proposé dans le **Problème 2** du TD 1. Vérifier les résultats énoncés en cours, c'est-à-dire :
   - Que la suite u_n diverge.
   - La valeur IEEE 754 de 1/3.

     Que constatez-vous par rapport à la valeur calculée en cours ? Pouvez-vous l'expliquer ?
2. Changer la valeur `float` par `double` dans la fonction `u_n`. Que constatez-vous ? Calculer l'erreur $\delta_{\max}$ commise dans ce cas-là. Combien vaut l'erreur Δ_{42} commise par la machine pour $n = 42$ dans ce cas ? Le résultat théorique de Δ_n et celui donné par la machine sont-ils cohérents ?

Multiplication et opération de décalage

L'opérateur $x \ll k$ est l'opérateur de décalage à gauche qui l'ajout k bits à 0 en poids faible de x. On rappelle que :

$$x \ll k = x \times 2^k$$

1. L'opération $x \gg k$ permet de réaliser la division entière de x par 2^k, mais cette opérateur est également très utilisé pour défiler un à un les bits d'un nombre x. On s'arrête alors lorsque x est égale à 0.

   Écrire un algorithme `int log_2(int x)` utilisant cet opérateur pour calculer la valeur $\lfloor \log_2(x) \rfloor + 1$ qui correspond au nombre de bits utilisés pour représenter x.

2. L'opérateur logique `&` est très utilisé en réseau où il sert à élaborer des masques permettant de sauvegarder des bits à certains emplacements souhaités. Par exemple, soient les masques :

   ```
   int masque1 = 0b111;
   int masque2 = 0b1001;
   ```

   Combiner avec l'opérateur `&` , le `masque1` permet de sauvegarder les 3 derniers bits de poids faibles d'un nombre x. Le `masque2` lui sauvegarde les bits 0 et 3.

   (a) Que donne les opérations `7 & masque1`, `8 & masque1`, `8 & 7`, `8 & 2`, `8 & 5`, `7 & 1` et `8 & 1`. Expliquer brièvement ces résultats.

   (b) Que permet de faire le masque `int masque3 = 0b010110` ? Le masque `int masque4 = 3` ? Quel masque doit-on appliquer pour récupérer les bits 2 et 1 ? Comment peut-on récupérer les 3 premiers bits de poids fort ?

3. En utilisant uniquement l'opération `&`, comment réalisez-vous l'opération modulo (`%`) 2 ?

4. Comment effectuer $x \times 5$ à l'aide des opérateurs `<<`, `&` et `+` ? Comment faire avec $x \times 7$?

5. Écrire un algorithme `int mul(int x, int n)` permettant de réaliser la multiplication $x \times n$ à l'aide du nombre minimal d'opérateurs `<<` et `+` pour tout $n \in \mathbb{N}^*$.

Half precision en IEEE 754

Le format half precision est un format IEEE 754 utilisé dans les systèmes légers dotés de peu de mémoire. Celui-ci permet d'implémenter des nombres flottants à l'aide de 2 octets (16 bits) seulement. Celui-ci est décrit dans le problème 4 du TD 1. On rappelle le codage d'un `half` :

- 1 bit de signe
- 5 bits d'exposant
- 10 bits de mantisse
- Un biais $\varepsilon = 15$

Récupérer la classe `Half.java` sur Celene. Celle-ci permet de représenter un nombre au format IEEE 754 sur 16 bits.

1. Quel nombre obtenez-vous en déclarant

   ```
   Half h = new Half(0, 0b10010, 0b1001000000);
   ```

   Vérifier le résultat également par le calcul à l'aide de la formule de décodage.

2. Écrire un nouveau constructeur `Half(float f)` qui convertie un `float` en Half precision. Pour se faire, on pourra récuperer les valeurs de signe, exposant et mantisse du nombre flottant f , à l'aide de masques, puis les recoder dans notre Half. On respectera les consignes écrites dans la méthode concernant les arrondis vers l'infini ou vers 0.

Calcul de $\sqrt{2}$

On cherche à représenter sur un half la valeur de $\sqrt{2}$. Pour se faire, on va utiliser l'algorithme de Newton et ré-utiliser la classe Half précédemment créée.

1. Donner la suite $(x_n)_{n\in\mathbb{N}}$ associée à l'algorithme de Newton permettant de calculer $\sqrt{x}$, pour tout $x \in \mathbb{R}^+$.
2. À propos du codage de la valeur de $\sqrt{2}$:
   (a) Quel est le pas de différence entre un nombre x et son plus proche voisin $x^{\pm}$ sur un Half precision ? Expliquer pourquoi.
   (b) Déterminer le rang k tel que $|x_{k+1} - x_k| < 2^{-10}$. On pourra écrire un programme ou le démontrer de manière formelle.
3. Déterminer la mantisse M, l'exposant E et le bit de signe s permettant de représenter $\sqrt{2}$ sous un Half à l'aide du terme x_k de la suite de Newton.

Logique et mémoire

On précise que dans ce chapitre les valeurs Vrai et Faux sont représentées par les constantes 1 et 0 contenues dans des entiers int.

Implémentation d'un additionneur

On souhaite ré-implémenter ici l'addition de deux nombres entiers ***uniquement*** à l'aide des opérateurs logiques bit-à-bit et de décalages. Les opérations $+, -$ et $\times$ sont exclues.

1. Écrire un programme `int retenue(int x, int y, int c)` correspondant au circuit logique de la retenue pour l'addition de deux bits x et y et d'une retenue c.
2. Écrire un programme `int sortie(int x, int y, int c)` correspondant au circuit logique de la sortie d'un bit z pour l'addition de deux bits x et y et d'une retenue c.
3. Soit le programme suivant :

```
int fonction(int x) {
    int i = 1;
    for(int y = x; (y & 1) == 1; y >>= 1)  {
       i = (i << 1) | 1;
    }
    return x ^ i;
}
```

   Que fait ce programme ? Expliquez-le.

4. Écrire un programme `int sortie(int X, int Y)` qui réalise l'addition de deux nombres X et Y. On pourra éventuellement utiliser le programme présenté question 3.

Clé de parité

Soit un nombre entier $x = \langle x_n ... x_2 x_1 \rangle_2$. On rappelle que la clé de parité (ou checksum) $P(x)$ correspond au nombre de bits à 1, compté modulo 2, dans la représentation binaire de x.

$$P(x) = \bigoplus_{i=1}^{x} x_i$$

1. Écrire le programme `int cle_parite(int x)` permettant de calculer et d'ajouter la clé de parité en bit de poids faible au nombre x.
2. On s'intéresse ici à la méthode de codage φ par parité longitudinale et transversale étudiée dans le **Problème 3** du TD 4. Cette méthode de correction logique permet de contrôler l'intégrité des données stockées en mémoire.

   On rappelle que les caractères ASCII sont codés sur 7 bits. Ainsi, un texte $x \in \text{ASCII}^n$ peut être représenté par une matrice X de taille $(n \times 7)$ où chaque ligne est le codage binaire du caractère x_i. Le texte $x =$ "chat" se représente tel que :

$$X = \begin{pmatrix} \langle \text{'c'} \rangle_2 \\ \langle \text{'h'} \rangle_2 \\ \langle \text{'a'} \rangle_2 \\ \langle \text{'t'} \rangle_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

(a) Écrire un programme `void print_textMatrix(String x)` qui imprime en console la forme matricielle d'un texte x donné en entrée. Que donne votre programme pour l'entrée $x =$ "J'ai faim !".

(b) Écrire un programme `int[] encodage(String x)` qui encode une chaîne de caractères ASCII selon la méthode par parité longitudinale et transversale et redonne un tableau d'entiers `tab` où `tab[i]` correspond au codage décimal de la ligne binaire i de $\varphi(X)$.

Localité des données et multiplication de matrices

Cet exercice reprend différentes questions du **Problème 4** du TD 4.
Une matrice est conceptuellement un tableau de nombres à 2 dimensions, mais en interne un tableau ne peut avoir qu'une seule dimension. Pour cela, il y a 2 techniques :

- Pour la 1ère dimension (par exemple n° de ligne) on stocke un tableau d'adresses. Chaque adresse renvoie au tableau des éléments de la ligne (éléments numérotés par leur colonne). On a donc un tableau d'adresses de tableaux de nombres. Technique du Java, Caml et Python. On écrit alors `M[i][j]`.
- On stocke tous les éléments de la matrice dans un seul grand tableau `t`, et l'élément M_{ij} d'une matrice s'obtient par `M[formule en i et j]`. Technique du C et du Fortran, moins souple, plus efficace.

Cette dernière technique admet 2 variantes, selon le sens dans lequel on représente les données. Voici par exemple la matrice :

$$M = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

- Convention du langage C : les éléments d'une même ligne sont toujours consécutifs, et on stocke les lignes à la suite (donc les données sont stockées dans l'ordre habituel de l'écriture). Exemple : M se stocke $(abcdefghi)$
- Convention du Fortran : les éléments d'une même colonne sont toujours consécutifs, et on stocke les colonnes à la suite. Exemple : M se stocke $(adgbehcfi)$.

On adopte les conventions du C.

1. Soit M une matrice à n lignes et p colonnes, dont les indices commencent à 0 (càd : $0 \leq i < n$ et $0 \leq j < p$). Donner la formule `M[...]` qui fournit M_{ij}.
2. Écrire une classe :

```
class Matrice {
   int n, p; // nb lignes, nb colonnes
   double M[]; // une seule dimension
}
```

et écrire ce constructeur et ces méthodes :

- `Matrice(int n, int p)` qui retourne la matrice nulle (matrice nulle).
- `double get (int i, int j)` qui permet d'obtenir le coefficient M_{ij}.
- `void set (int i, int j, double x)` qui met à jour le coefficient $M_{ij} = x$.

- `Matrice t()` calcule et retourne la transposée de M.

3. Soient A une matrice à n lignes et q colonnes et B une matrice à q lignes et p colonnes, on note $C = AB$. Pour calculer C, on calcule chaque produit contractant :

$$C_{ij} = \sum_{k=0}^{q-1} A_{ik} B_{kj}$$

   Écrire un algorithme `coeff(Matrice B, int i, int j) : A.fois(B, i, j)` qui calcule le produit contractant C_{ij}.

4. Écrire un algorithme `Matrice fois(Matrice B): A.fois(B)` qui multiplie deux matrices sous forme de tableau à une dimension selon la convention du C. On lèvera les exceptions nécessaires si la multiplication ne peut pas être effectuée.

5. Dans chaque produit contractant de l'algorithme précédent, que dire des accès mémoire concernant A ? Concernant B ? Ce calcul exploite t-il pleinement les principes de localité spatiale des caches ? Expliquer pourquoi ?

6. Au lieu de calculer AB, quel calcul similaire à AB faudrait-il effectuer pour accélérer les accès mémoire en lecture ?

7. Écrire un algorithme `Matrice fois_opt(Matrice B)` qui multiplie deux matrices avec le maximum d'efficacité mémoire en s'appuyant sur les principes de localité des caches.

8. Tester votre algorithme sur la multiplication de deux matrices de taille 2 000 × 2 000. Combien de temps prend la multiplication ? Tester également avec la multiplication matricielle selon la méthode de représentation classique de Java. Que pouvez-vous conclure ?