Ecole polytechnique de l'Université de Nantes
Travaux dirigés E.T.N. 3
Méthodes numériques

Vincent GOURET

14 février 2017

TD machine 1 : représentation machine des réels

Ce TD a pour objectifs :

1. de déterminer la place mémoire utilisée par les types courants du langage C, et de concrétiser le format machine adopté pour la représentation des réels.
2. d'éclairer le concept de précision machine, étroitement lié au format machine utilisé.
3. d'expérimenter la non-associativité d'opérations qui mathématiquement le sont, mais pas lors de calculs sur ordinateur. L'approche est ici statistique (sous une forme très basique), sous l'interprétation fréquentielle des probabilités.
4. de quantifier le temps d'exécution des opérations algébriques de base et des opérations plus complexes comme les lignes trigonométriques ou l'extraction de racines carrées, en utilisant des réels. Le format machine des réels influe peu sur ces temps, par contre certaines opérations sont plus coûteuses que d'autres.

A l'issu de ce TD, vous devriez connaître quantitativement les caractéristiques de la machine que vous utilisez, vis-à-vis des deux concepts centraux en méthodes numériques que sont le coût (temps d'exécution, mémoire utilisée, ...) et la précison.

Tous les programmes sont écrits en langage C et exécutés sous un environnement Linux. On rappelle que lorsque le source est fourni, il faut le recueillir sur Madoc. On rappelle aussi la commande pour compiler un source : `cc -o prog prog.c -lm`. Ici, `prog` désigne le fichier exécutable à obtenir et `prog.c` est le fichier source. L'option `lm` est nécessaire à la compilation quand on utilise les fonctions mathématiques (utilisation du fichier header `math.h`). Pour terminer cette courte introduction, un mot sur l'évaluation : il vous est demandé de déposer sur Madoc un fichier au format PDF contenant le compte rendu de ce TD dans la semaine qui suit votre séance.

1.1 Taille des types courants en C

En utilisant le source `TailleTypes.c`, déterminez les nombres d'octets nécessaires pour chacun des types courants en C.

1.2 Représentation des réels en machine

En utilisant le source `ShowFloat.c`, vérifiez la représentation des réels pour la simple précision (pensez à utiliser l'exemple du cours : $-28,5$, dont la valeur est connue). En étudiant le source, votre machine est-elle little-endian ou big-endian ?

1.3 Précision machine

Ecrire un programme en C permettant de calculer la précision machine suivant l'algorithme suivant :

```
x = 1
tant que 1+x > 1 faire : x = x/2
afficher x
```

Comparez les valeurs obtenues avec celles correspondant à la simple et double précision. **Attention :** il est indispensable que le test `1+x>1` porte sur des réels du type considéré : il faut donc placer dans une variable du type concerné le résultat de `1+x` avant d'effectuer le test.

1.4 Associativité en péril

Analysez le source `AssocFloat.c` et expliquez sa fonction. En variant le nombre d'itérations donné au départ sur la ligne de commande, estimez la fréquence du phénomène de non-associativité. Modifiez ensuite le source pour considérer la double précision.

Bonus : En vous inspirant de ce source, écrivez un programme montrant que la distributivité n'est pas toujours vérifiée.

1.5 Vitesse d'exécution des opérations

En utilisant le source `Flop.c`, estimez le nombre d'opérations usuelles (addition, soustraction, multiplication et division) exécutées par seconde par votre machine pour la simple précision, puis la double précision et enfin la précision étendue. Modifiez ensuite le programme pour estimer le nombre d'autre opérations : racine carré et lignes trigonométriques. Etablissez une synthèse de vos résultats sous la forme d'un tableau à deux entrées (type de précision et type d'opération) donnant les durées des opérations (on considérera comme négligeables les temps nécessaires à la lecture et l'écriture des variables).

TD machine 2 : comptage des opérations d'un algorithme

Ce TD a pour objet la détermination expérimentale et théorique du nombre d'opérations demandées par un algorithme de calcul déterministe (les nombres d'opérations ne dépendent que de la taille du problème posé, pas des données qui le particularisent). L'approche expérimentale consiste à implanter l'algorithme en remplaçant les opérations dont on veut déterminer le nombre par un comptage. L'objectif est que les étudiants comprennent l'origine polynomiale de la complexité des algorithmes envisagés ; l'imbrication de boucles for détermine le degré de ladite complexité polynomiale. On rappelle que lorsque le source est fourni, il faut le reccueillir sur Madoc. Le compte-rendu sera à déposer sur Madoc au format PDF exclusivement durant la semaine qui suit votre séance.

2.1 Expérimentation : calcul du nombre d'opérations par implentation machine

On considère l'algorithme 1, permettant de résoudre un système linéaire de n équations à n inconnues. On se propose d'évaluer le nombre d'opérations nécessaires à cet algorithme de la façon suivante : on implante l'algorithme concrêtement en utilisant deux compteurs, l'un pour les additions/soustractions et l'autre pour les multiplications/divisions ; les lignes de l'algorithme effectuant des affectations sont élininées et celles effectuant des opérations sont remplacées par des instructions permettant le comptage des dites opérations. L'algorithme 1 devient ainsi l'algorithme 2.

Implantez l'algorithme 2 en C et déterminez ainsi le nombre d'opérations nécessaires pour $n \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 100, 200\}$. Vous noterez les résultats obtenus sous forme d'un tableau.

2.2 Théorie : détermination du nombre d'opérations. Formule de Faulhaber

2.2.1 Boucle for simple

On considére l'algorithme 3. Déterminez l'expression de N en fonction de n d'un point de vue théorique et vérifier cette expression à l'aide d'un programme en C implantant cet

Algorithme 1 Méthode de Gauss

```
1: for i = 1 to n − 1 do
2:   for j = i + 1 to n + 1 do
3:     w_{i,j} ← w_{i,j}/w_{i,i}
4:   end for
5:   w_{i,i} ← 1
6:   for j = i + 1 to n do
7:     for k = i + 1 to n + 1 do
8:       w_{j,k} ← w_{j,k} − w_{i,j} × w_{j,k}
9:     end for
10:    w_{j,i} ← 0
11:  end for
12: end for
13: w_{n,n+1} ← w_{n,n+1}/w_{n,n}
14: w_{n,n} ← 1
15: for i = n − 1 to 1 do
16:   for k = i + 1 to n do
17:     w_{i,n+1} ← w_{i,n+1} − w_{i,k} × w_{k,n+1}
18:   end for
19: end for
```

Algorithme 2 Comptage pour la méthode de Gauss

```
1: N_a ← 0
2: N_m ← 0
3: for i = 1 to n − 1 do
4:   for j = i + 1 to n + 1 do
5:     N_m ← N_m + 1
6:   end for
7:   for j = i + 1 to n do
8:     for k = i + 1 to n + 1 do
9:       N_m ← N_m + 1
10:      N_a ← N_a + 1
11:    end for
12:  end for
13: end for
14: N_m ← N_m + 1
15: for i = n − 1 to 1 do
16:   for k = i + 1 to n do
17:     N_m ← N_m + 1
18:     N_a ← N_a + 1
19:   end for
20: end for
21: return N_a, N_m
```

algorithme en prenant des valeurs concrêtes pour n : $n \in \{2, 10, 20, 30\}$.

Algorithme 3 Algorithme simple

```
1: N ← 0
2: for i = 1 to n − 1 do
3:    N ← N + 1
4: end for
5: return N
```

2.2.2 Boucles for imbriquées

On considère maintenant l'algorithme 4. Déterminez l'expression de N en fonction de n d'un point de vue théorique et vérifier cette expression à l'aide d'un programme en C implantant cet algorithme en prenant des valeurs concrêtes pour n : $n \in \{2, 10, 20, 30\}$.

Algorithme 4 Deux boucles imbriquées

```
1: N ← 0
2: for i = 1 to n − 1 do
3:    for j = 1 to n − 1 do
4:       N ← N + 1
5:    end for
6: end for
7: return N
```

Soit maintenant l'algorithme 5. Déterminez l'expression de N en fonction de n d'un point de vue théorique et vérifier cette expression à l'aide d'un programme en C implantant cet algorithme en prenant des valeurs concrêtes pour n : $n \in \{2, 10, 20, 30\}$. Vous pourrez utiliser la formule suivante :

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Algorithme 5 Deux boucles imbriquées et liées

```
1: N ← 0
2: for i = 1 to n − 1 do
3:    for j = i to n − 1 do
4:       N ← N + 1
5:    end for
6: end for
7: return N
```

2.2.3 Boucles for imbriquées à nombre variable d'itérations

On considère de nouveau l'algorithme 2. Donnez les expressions théoriques pour les entiers N_a et N_m en fonction de la taille du problème n. Vous pourrez utiliser la formule suivante :

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

Comparez votre formule aux résultats obtenus dans la partie expérimentale.

2.2.4 Formule de Faulhaber

En 1631, J. Faulhaber publie la formule générale de la somme des puissances des n premiers entiers (la formule qui suit est sous sa forme moderne) :

$$\sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{k=1}^{m+1} (-1)^{\delta_{k,m}} \binom{m+1}{k} B_{m+1-k} n^k$$

où $\delta_{k,m}$ désigne le symbole de Kronecker (égal à 1 si $k = m$ et 0 sinon), $\binom{m+1}{k}$ désigne le coefficient binomial bien connu :

$$\binom{m+1}{k} = \frac{(m+1)!}{k!(m+1-k)!}$$

et B_{m+1-k} désigne le $m+1-k$-ième nombre de Bernouilli, qui peuvent être définis à l'aide de la récurrence suivante :

$$\sum_{k=0}^{n} \binom{n+1}{k} B_k = 0$$

avec comme condition initiale $B_0 = 1$. On montre que dès lors que n est impair et strictement supérieur à 1, $B_n = 0$. Utilisez la formule de Faulhaber pour retrouver les expressions des nombres d'opérations dans tous les exercices précédents.

Quelle structure d'algorithme demande un nombre d'opérations mettant en jeu la somme des cubes des n premiers entiers ? Implantez l'algorithme proposé et exprimez en utilisant les formules de Faulhaber le nombre d'opérations en fonction de n. Comparez votre expression aux résultats obtenus par la machine.