Introduction à l'arithmétique des ordinateurs

Sylvie Boldo

Introduction

La puissance de nos ordinateurs augmente chaque mois et on peut paralléliser un même programme sur plusieurs ordinateurs. Tout cela permet de faire des calculs de plus en plus compliqués mais on peut s'interroger sur la validité du résultat de tels calculs [2].

De retentissants échecs tels que le bug du Pentium ou l'explosion d'Ariane 5 ont démontré qu'il ne fallait pas croire aveuglément aux résultats d'un calcul sur ordinateur. L'ordinateur ne peut calculer parfaitement et le résultat fourni ne doit pas être pris comme parole d'évangile.

En effet, l'ordinateur n'a qu'une mémoire finie. Il ne peut donc pas manipuler des nombres réels infiniment précis mais ce qu'on appelle des nombres à virgule flottante.

1 Les nombres à virgule flottante

En machine, un nombre à virgule flottante n'est qu'une suite de bits (0 ou 1). Ces bits se répartissent en 3 champs : le signe, la fraction et l'exposant comme montré en figure 1.

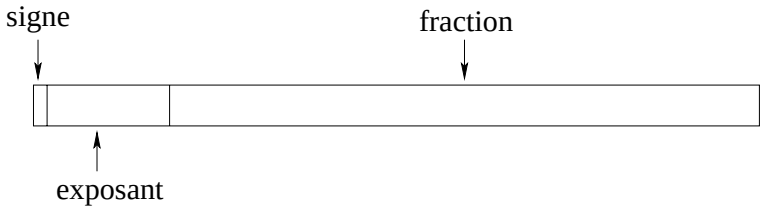

FIG. 1 – *Les champs d'un nombre à virgule flottante*

La plupart des processeurs actuels respectent la norme IEEE-754 [7] qui impose notamment deux formats de nombres à virgule flottante appelés simple et double précision. Un nombre en simple précision est composé de 32 bits : 1 pour le signe, 23 pour la fraction et 8 pour l'exposant. Un nombre en

simple précision est composé de 64 bits : 1 pour le signe, 52 pour la fraction et 11 pour l'exposant.

Notons s le signe, E l'exposant et $f = b_1 b_1 \dots b_p$ la fraction du nombre à virgule flottante avec $b_i = 0$ ou 1. L'exposant est biaisé, ce qui signifie que le vrai exposant e vaut $E + biais$. Sauf cas particuliers, la valeur du nombre à virgule flottante est alors $(-1)^s \times 1.f \times 2^e$ où $1.f = 1 + \sum_{i=1}^{p} b_i 2^{-i}$.

En plus de tels nombres (dits normalisés), on peut représenter des nombres plus petits (dits dénormalisés), $\pm\infty$, ± 0 et NaN. Les deux zéros assurent une cohérence des signes de l'inverse : $\frac{1}{+0} = +\infty$ et $\frac{1}{-0} = -\infty$. NaN, qui signifie *Not A Number*, est le résultat d'opérations telles que $\sqrt{-1}$, $\infty - \infty$, $\frac{\infty}{\infty} \dots$

Comme le résultat exact d'une opération ne peut pas forcément tenir dans le format voulu, la norme IEEE-754 définit quatre modes d'arrondi. L'arrondi vers $-\infty$ ($\bigtriangledown(x)$) est le nombre à virgule flottante le plus grand inférieur ou égal à x ; l'arrondi vers $+\infty$ ($\bigtriangleup(x)$) est le plus petit supérieur ou égal à x ; l'arrondi vers 0 ou troncature ($\mathcal{Z}(x)$) vaut $\bigtriangledown(x)$ si $x \geq 0$ et $\bigtriangleup(x)$ sinon et l'arrondi au plus proche ($\mathcal{N}(x)$) est le plus proche de x. Si deux nombres à virgule flottante sont à égale distance de x, c'est celui dont la mantisse est paire (qui finit par un 0) qui est choisi.

Pour toutes les opérations de base ($+, -, \times, /, \sqrt{}$), la norme impose que le résultat renvoyé soit le même que si l'on avait calculé avec une précision infinie et arrondi ensuite. On ne peut pas avoir un meilleur résultat qui soit représentable dans le format voulu.

Ces propriétés sont très fortes et permettent de faire des preuves sur les calculs flottants comme en [4]. Néanmoins, bien que l'arrondi soit correct, le résultat n'est pas le résultat mathématique exact, ce qui peut créer des résultats absurdes.

2 Quelques exemples de problèmes

L'algorithme Certains algorithmes ont la particularité d'être stables : si l'on change un peu les données initiales, le résultat ne varie que très peu. Il est préférable d'utiliser ces algorithmes car ils donneront presque à coup sûr un bon résultat une fois implanté.

Le programme Le programmeur peut oublier que certains évènements exceptionnels (appelés exceptions) peuvent se produire. Ce sont par exemple les divisions par zéro ou les dépassements de capacité. Ainsi en double précision, pour $x = 2^{60}$ et $y = 1$, le calcul de $\frac{1}{(x+y)-x}$ fait une division par zéro, ce qui est contraire à l'intuition puisque y est non nul !

Le processeur Si l'unité flottante du processeur présente des dysfonctionnements, il est certain que certains calculs ne donneront pas le bon résultat. Ce fut le cas du fameux "bug du Pentium" où certaines divisions n'étaient pas précises du tout.

3 Quelques solutions

Les solutions sont nombreuses [1] et parfois originales comme l'utilisation d'un arrondi aléatoire. Voici quelques autres possibilités.

Certains problèmes sont irréparables : en effet, si l'unité flottante ne donne pas le bon résultat, il n'y a pas grand-chose à faire. Une solution est de prouver de façon certifiée que les opérations du processeur sont correctes comme l'ont fait Harrison (Intel) et Russinoff (AMD).

Une solution à beaucoup de problèmes serait de calculer avec plus de précision [3, 6]. Cela permet de retarder les effets néfastes des dépassements de capacité et des algorithmes instables.

Une solution utilisée mais coûteuse est le calcul par intervalles [5]. Le résultat est alors un intervalle de nombres à virgule flottante qui contient le résultat exact. Cela se fait par exemple en utilisant les propriétés des modes d'arrondi IEEE-754.

Conclusion

Le résultat d'un calcul numérique complexe sur ordinateur est donc tout-à-fait susceptible d'être faux sans que l'utilisateur en soit prévenu. De nombreuses recherches sont menées pour éviter des désagréments et les réponses sont diverses et originales. Il est donc possible en utilisant certaines techniques d'être sûr du résultat de son calcul.

La tendance actuelle est en effet à la garantie du résultat : on veut pouvoir faire confiance aveuglément à la machine et ne pas se poser de questions,

même si le calcul doit prendre plus de temps. Un moyen de donner des garanties à l'utilisateur est de faire des preuves mais ces preuves sont sujettes à l'erreur et ne peuvent donc être totalement rassurantes. On peut alors faire vérifier ces preuves par des assistants de preuves qui vérifieront chaque cas et chaque détail de la preuve avant de l'accepter. On a alors une garantie nettement plus forte du résultat.

Références

[1] *Marc Daumas et Jean-Michel Muller (éditeurs)*, Qualité des calculs sur ordinateur : vers des arithmétiques plus fiables. Masson (1997)

[2] *David Goldberg*, What every computer scientist should know about floating point arithmetic. ACM Computing Surveys (1991)

[3] *Guillaume Hanrot, Vincent Lefèvre, Fabrice Rouillier et Paul Zimmermann*, The MPFR library. `www.mpfr.org`. Version 2001.

[4] *Michèle Pichat*, Contributions à l'etude des erreurs d'arrondi en arithmétique à virgule flottante. Thèse (1976)

[5] *Nathalie Revol et Fabrice Rouillier*, The MPFI library. version 2001, `http://www.ens-lyon.fr/~nrevol`. (2001)

[6] *Jonathan R. Shewchuk*, Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. Discrete and Computational Geometry (1997)

[7] *David Stevenson et al.*, IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE (1985)

Sylvie Boldo
Laboratoire de l'Informatique du Parallélisme
UMR 5668 CNRS-INRIA-ENS Lyon
École Normale Supérieure de Lyon
46, Allée d'Italie
69364 Lyon Cedex 07
France
`Sylvie.Boldo@ens-lyon.fr`
http://www.ens-lyon.fr/~sboldo