Des ordinateurs capables de calculer plus juste

Additionner 3 nombres quelconques sans erreur peut devenir mission impossible pour un ordinateur. Afin de pallier ce manque de fiabilité, les informaticiens inventent une nouvelle arithmétique. Indispensable pour construire un pont ou une aile d'avion.

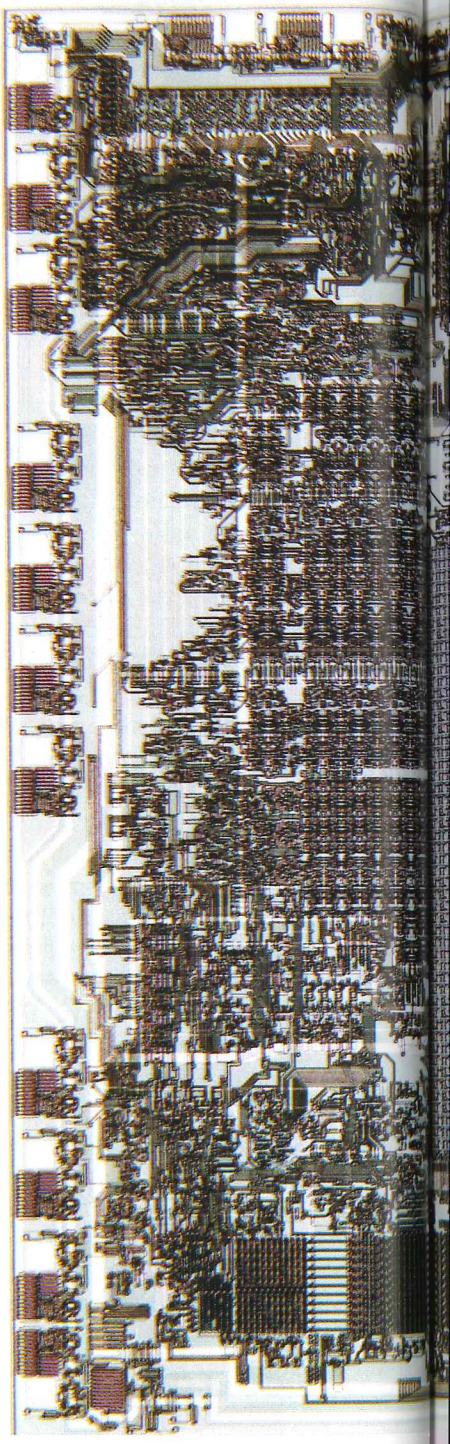

PAR **Sylvie Boldo,**
chargée de recherche Inria à Orsay,
ET **Jean-Michel Muller,** directeur de recherche CNRS au laboratoire de l'informatique du parallélisme à Lyon, l'un des lauréats du prix *La Recherche* 2013.

L'essentiel

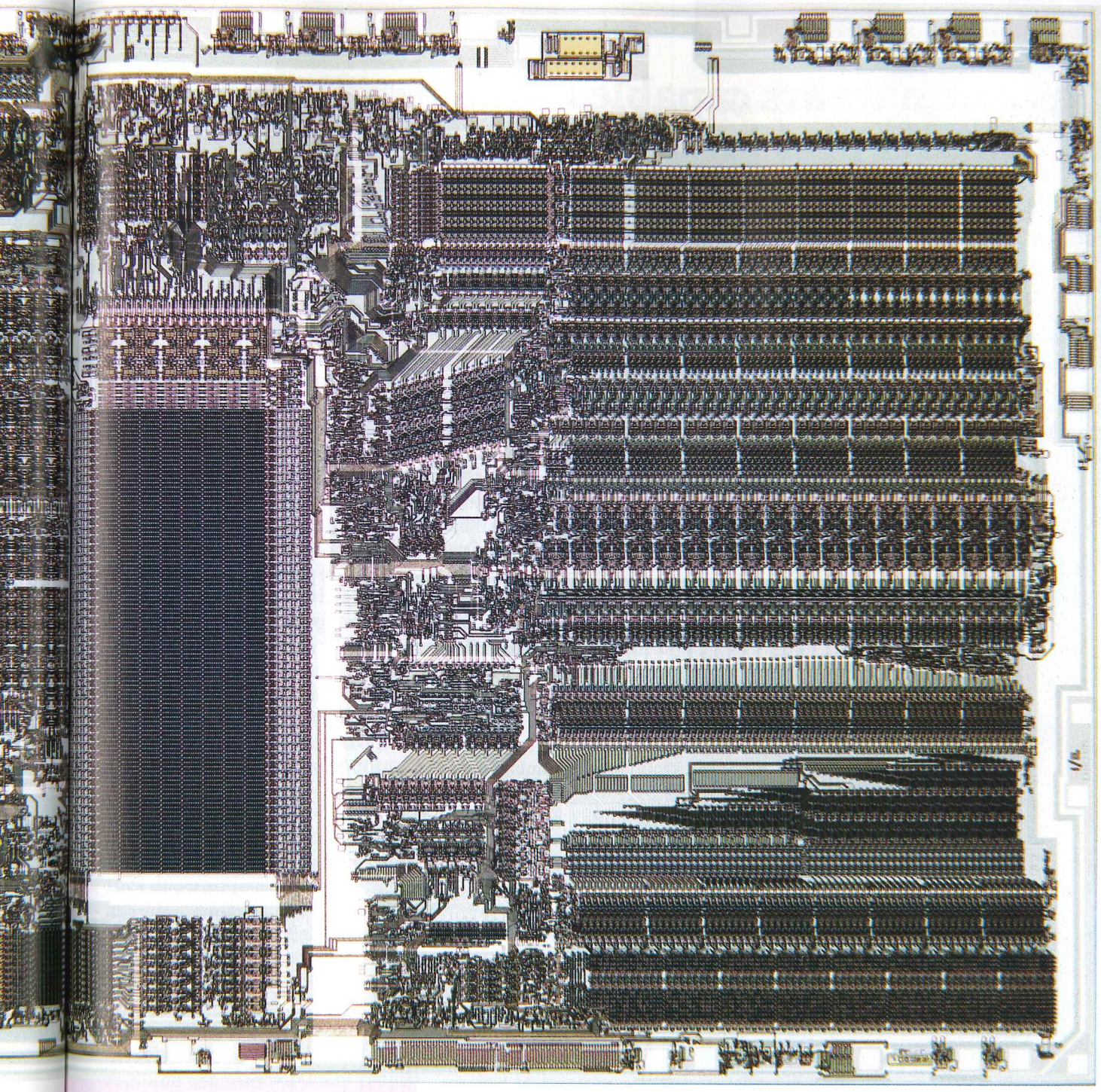
- LES ORDINATEURS, ayant une précision finie, ne peuvent pas calculer de manière exacte.
- POUR AUGMENTER la précision des calculs et ne pas obtenir de résultats aberrants, il faut faire les calculs selon des règles d'arrondis spécifiques.
- CES RÈGLES permettent de construire des algorithmes qui compensent partiellement les erreurs commises.

Les ordinateurs ne savent pas calculer. Ou du moins, ils ne peuvent pas calculer exactement, sans erreurs. Dès lors, les résultats des calculs numériques effectués avec une machine ne peuvent être que des approximations. Et si ces approximations sont mal faites, ou s'il y a des erreurs de programmation, on aboutit parfois à des résultats étranges.

Ainsi, dans les années 1960, sur certains ordinateurs IBM 360, lorsqu'on attribuait dans un programme Fortran la valeur 14,0 divisé par 7,0 à la variable I, celle-ci devenait par la suite égale à 1. À la fin des années 1970, sur certains ordinateurs de la société Cray, on pouvait déclencher un dépassement de capacité en multipliant un nombre par un. Ou encore, l'algorithme de division de la première version du processeur Pentium de la société Intel lancé en 1994 était incorrect : la division de 8 391 667 par 12 582 905 donnait 0,666 869... au lieu de 0,666 910. On pourrait multiplier les exemples.



Le comportement étrange de ces systèmes anciens est certes anecdotique, mais il illustre la nécessité de concevoir pour les ordinateurs une arithmétique qui calcule bien... ou le moins mal possible. Sans comportement prévisible, pas de calculs numériques fiables, ce qui est problématique pour la recherche scientifique, mais aussi si l'on veut construire un pont solide ou une aile qui permette à un avion de voler.



Ce coprocesseur de calcul Intel 8087 a été le premier, en 1980, à être largement diffusé et à offrir l'« arrondi correct ». Les nombres codés par la machine sont arrondis selon une convention prédéfinie, mais quand on les utilise pour une opération, le résultat est celui que l'on trouverait en effectuant celle-ci sur les nombres réels, puis en appliquant la convention d'arrondi. © INTEL

En 1985, une norme sur la représentation des nombres en machine et leur calcul a été établie (IEEE-754). Cette norme, révisée en 2008, spécifie la manière dont les machines doivent calculer pour obtenir des calculs fiables (lire « Une norme qui s'impose petit à petit », p. 49). Elle s'appuie sur des algorithmes élaborés et testés par les chercheurs depuis plusieurs décennies. La version 2008 de cette norme, qui contient les « briques de base » permettant de faire du calcul complètement spécifié, n'est pas encore universellement utilisée, mais on peut parier que d'ici à quelques années, la plupart des logiciels utilisant l'analyse numérique s'y conformeront.

Ces briques de base consistent en des algorithmes qui permettent de faire des opérations arithmétiques de manière précise. De récents progrès permettent par exemple de mieux comprendre les erreurs effectuées >>>

Des ordinateurs capables de calculer plus juste

››› lorsqu'on utilise un algorithme qui combine addition et multiplication, ou encore de faire des additions, des multiplications et des évaluations de polynômes précis avec des nombres complexes [1]. Pour illustrer quelques principes de ces algorithmes, nous allons utiliser l'opération la plus simple : l'addition. Après avoir rappelé la manière dont les ordinateurs représentent les nombres, nous montrerons comment rendre les calculs le plus précis possible dans le cadre de l'addition de plusieurs nombres. La clé du succès consiste ici à contrôler en permanence l'erreur que l'on fait sur le calcul.

De nombreuses applications scientifiques utilisent de très grands nombres et de très petits. Par exemple le volume du Soleil est de 1 412 000 000 000 000 000 000 000 000 mètres cubes. Représenté ainsi, un tel nombre n'est pas facile à manipuler (il faut compter le nombre de chiffres pour avoir une idée de son ordre de grandeur). De manière similaire, la masse de l'électron est de 0,000 000 000 000 000 000 000 000 000 910 9 kilogramme, ce qui n'est pas non plus une représentation commode.

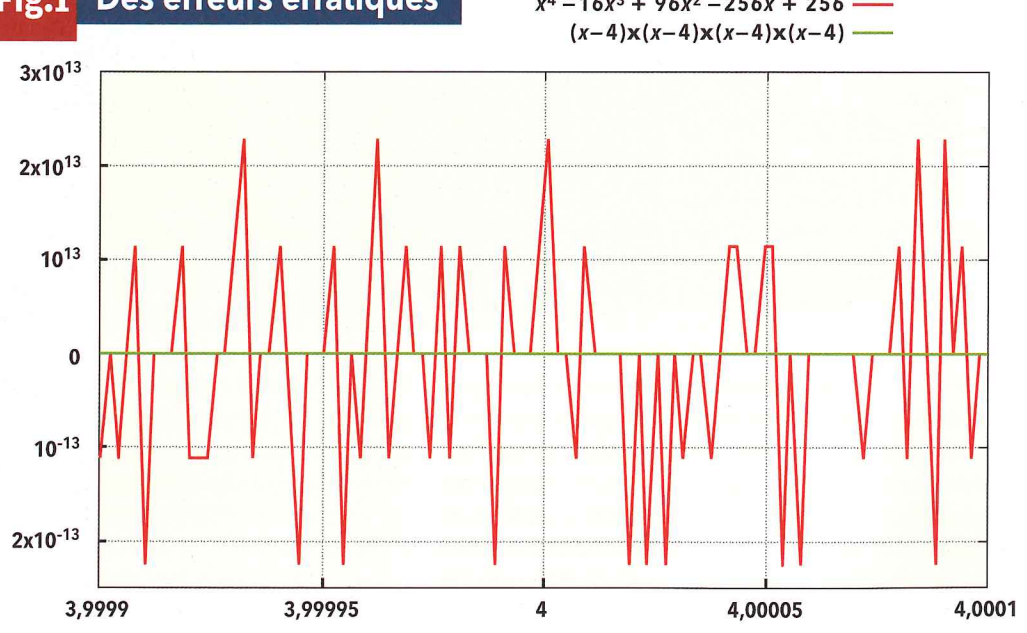
Virgule flottante. Afin de manipuler de tels nombres, on utilise en général la « notation scientifique ». Ses prémices remontent à Descartes, qui proposa en 1637 dans sa *Géométrie* la notation 10^n pour exprimer un nombre où 10 est multiplié n fois par lui-même. La notation scientifique en elle-même s'est répandue à partir du XIX^e siècle : tous les nombres peuvent s'écrire sous la forme $m \times 10^n$, où m est compris entre 1 et 10, et n est un entier. Dans cette notation, le volume du Soleil se note $1{,}412 \times 10^{27}$ mètres cubes et la masse de l'électron $9{,}109 \times 10^{-31}$ kilogramme, ce qui est plus aisé à manipuler.

Dans un ordinateur, la représentation la plus courante des nombres réels, que l'on appelle le « format virgule flottante » est une adaptation de la notation scientifique au contexte de calcul avec une précision finie. Un format virgule flottante est caractérisé par une base ß (qui en pratique vaut 2 ou 10), et par une précision p. Ce format étant fixé, un nombre virgule flottante x est écrit sous la forme $m_0, m_1 m_2 \ldots m_{(p-1)} \times ß^e$ où le nombre de p chiffres $m_0, m_1 m_2 \ldots m_{(p-1)}$ écrit en base ß est appelé la mantisse de x, et le nombre e, qui est un entier, est nommé exposant de x. Par exemple, si la précision est 5, le nombre 5 634 000 pourra s'écrire exactement en virgule flottante $5{,}634\,0 \times 10^6$, tandis que le nombre 0,087 698 7 ne pourra lui être qu'approché : $0{,}087\,698\,7 \approx 8{,}769\,9 \times 10^{-2}$.

Dans la plupart des ordinateurs, la base utilisée est 2. Le format binaire (en base 2) le plus utilisé, appelé « double précision » pour des raisons historiques, correspond à ß = 2 et p = 53. Dans ce format, le nombre virgule flottante le plus proche du nombre décimal 0,1 s'écrit en base 2 : 0,000 110 011 001 10 0 110 011 001 100 110 011 001 100 110 011 001 100 110 1. Sa valeur décimale est 0,100 000 000 000 000 005 551 115 123 125 782 702 118 158 340 454 101 562 5. Ainsi, lorsqu'on tape « 0,1 » sur un clavier d'ordinateur, c'est ce nombre que l'on obtient : on a commis une (légère) erreur avant même d'avoir commencé la première opération. Parfois, l'écriture des nombres dans un format « machine » fait ainsi déjà perdre de la précision.

Un format (base et précision) étant donné, comment effectuer des opérations ? Comment calculer la somme le produit, le quotient de deux nombres virgule flottante ?

Fig.1 Des erreurs erratiques



LA FONCTION $(x - 4)^4$ quand x est proche de 4 devrait peu varier autour de 0 (courbe en vert). Mais si on la calcule avec la formule développée $x^4 - 16x^3 + 96x^2 - 256x + 256$, la courbe (en rouge) est très irrégulière, à cause des erreurs d'arrondi du calcul en virgule flottante.

Une norme qui s'impose petit à petit

Avant que les normes de calculs n'apparaissent, les résultats que l'on pouvait obtenir étaient parfois fantaisistes. Dès 1980, le coprocesseur de calcul Intel 8087 fonctionnait avec un « arrondi correct », ce qui permettait de prédire les comportements de calculs. À partir de 1985, les constructeurs de processeurs – Intel, IBM, Apple, HP pour ne citer que les principaux – ainsi que des universitaires, dont William Kahan, professeur à l'université Berkeley, aux États-Unis, élaborent une norme pour le calcul en virgule flottante qui deviendra le standard IEEE-754 pour le calcul en virgule flottante. Révisée en 2008, cette norme, fondée sur les résultats des chercheurs, spécifie ce qui doit être fait quand on effectue une opération arithmétique, qu'il s'agisse d'addition, de soustraction, de division ou de racine carrée, les opérations de base à partir desquelles tout le calcul numérique peut être construit. Même si rien n'oblige un constructeur à la respecter, ne pas la suivre entraînerait une telle publicité négative que tous la respectent.

Le résultat d'une telle opération ne sera pas nécessairement un nombre virgule flottante, de sorte qu'il faudra l'arrondir. Par exemple, en base 5 et si la précision est 5, le produit des deux nombres $a = 1{,}2345 \times 10^2$ et $b = 5{,}4321 \times 10^{-2}$ vaut 6,705 927 45. Ce nombre ne pouvant pas être exactement représenté avec une mantisse de cinq chiffres, on sera amené à le remplacer par l'un des deux nombres virgule flottante qui l'encadrent, soit par $6{,}705\,9 \times 100$, soit par $6{,}706\,0 \times 100$. Lequel choisir ?

Fonction d'arrondi. Aux débuts de l'informatique, cette opération d'arrondi n'était pas clairement définie. Dans l'exemple précédent, on ne pouvait pas prédire lequel des deux nombres aurait été choisi. Et dans certains cas, cela aurait pu même être une tout autre valeur, d'où toutes les erreurs possibles qui s'ensuivent. La norme IEEE-754 spécifie maintenant clairement ce qui doit être fait lorsqu'on effectue une opération arithmétique à l'aide d'une machine.

Tout d'abord, l'utilisateur doit choisir une fonction d'arrondi (notée 0) qui, à un nombre réel x, fait correspondre >>>

Des ordinateurs capables de calculer plus juste

>>> un nombre virgule flottante. Il existe plusieurs choix [fig. 2]. Avec l'arrondi « vers le bas », $O(x)$ est le plus grand nombre virgule flottante inférieur ou égal à x. Avec l'arrondi « vers le haut », $O(x)$ est le plus petit nombre virgule flottante supérieur ou égal à x. Avec l'arrondi « vers le zéro », $O(x)$ est l'arrondi vers le bas de x si x est positif et l'arrondi vers le haut sinon. Enfin, avec l'arrondi « au plus près », $O(x)$ est le nombre virgule flottante le plus proche de x. Une convention permet de trancher si x est au milieu de deux nombres virgule flottante consécutifs : on arrondit vers celui dont le dernier chiffre de la mantisse est pair. Ensuite, chaque fois que l'on effectue une opération arithmétique, le résultat obtenu doit être celui qui l'on trouverait en effectuant tout d'abord l'opération exactement (avec une précision « infinie ») puis en appliquant la fonction d'arrondi à la valeur exacte.

Choix par défaut. Les opérations machines qui satisfont cette propriété sont dites « correctement arrondies ». Certains constructeurs avaient envisagé de mettre en place cette propriété dès les débuts de l'informatique, mais ce n'est qu'en 1980, avant même les débuts de la norme IEEE-754, qu'elle commence à être implantée, d'abord dans le coprocesseur Intel 8087.

Même si l'exigence d'arrondi correct a été un progrès considérable, elle ne suffit pas à éviter certains problèmes et résultats faux. Avant de l'expliquer sur un exemple, nous supposerons dans la suite de cet article que la fonction d'arrondi est l'arrondi au plus près (c'est le choix par défaut : sans instruction particulière, votre ordinateur utilisera cette fonction). Et on notera les opérations machines en entourant les symboles de l'opération exacte. Par exemple $\oplus$ sera l'« addition machine » et $\otimes$ la « multiplication machine ». Ainsi, pour des opérations machines correctement arrondies on aura : $a \oplus b = o(a + b)$ et $a \otimes b = o(a \times b)$.

Quels problèmes peuvent survenir ? Plaçons-nous dans le cadre d'une arithmétique de base 10, avec une précision égale à 10 – celle de nombreuses calculatrices de poche – et la fonction arrondie au plus près. Prenons le calcul simple suivant (qui ne comporte que deux opérations) : $10^{50} + 1 - 10^{50}$. La machine effectue *a priori* les opérations dans l'ordre naturel suggéré par la formule, c'est-à-dire que ce qu'elle calcule réellement est $(10^{50} \oplus 1) \ominus 10^{50}$. Comme le nombre virgule flottante le plus proche de $10^{50} + 1$ est 10^{50}, le résultat de l'addition sera 10^{50} ; la soustraction calculera alors $10^{50} - 10^{50}$ de sorte que le résultat obtenu sera 0, alors que le résultat correct est 1.

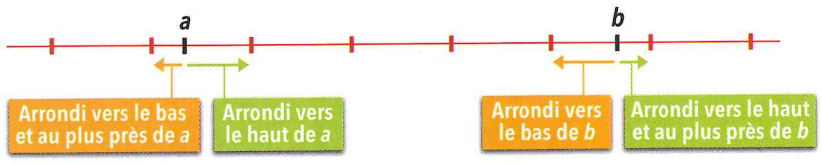
Ordre croissant. Pour appuyer notre propos et voir comment obtenir des calculs corrects à l'aide d'un ordinateur, prenons un exemple qui nous servira de « fil rouge » dans la suite de cet article : la simple somme de cinq nombres. Choisissons un grand nombre $M = 10^{18}$. Ce nombre est bien un nombre virgule flottante en décimal (mantisse 1 et exposant 18), mais c'est aussi un nombre virgule flottante en binaire (sa mantisse s'écrit 1,101 111 000 001 011 011 010 110 011 101 001 110 110 01 en base 2 et son exposant vaut 59). Vous pourrez donc essayer les algorithmes présentés aussi bien « à la main » ou sur votre calculatrice de poche (donc en base 10), que sur votre ordinateur (en base 2). La somme à calculer est : $S = M + M^2 + 1 - M - M^2$, dont le résultat exact est 1. Le calcul machine de M^2 est un peu délicat. En effet, 10^{36} est exact en décimal, mais pas en binaire (tout du moins tant que la précision est inférieure ou égale à 83, ce qui est le cas du format double précision pour lequel $p = 53$). Cela n'a pas grande importance ici, car la valeur calculée de M^2 (c'est-à-dire $M \otimes M$) est la même au début et à la fin de cette somme, de sorte que cette erreur n'intervient pas dans les résultats fautifs à venir.

Calculons d'abord cette somme de façon traditionnelle, de la gauche vers la droite : $S_1 = (((M \oplus M^2) \oplus 1) \ominus M) \ominus M^2 = ((M^2 \oplus 1) \ominus M) \ominus M^2 = (M^2 \ominus M) \ominus M^2 = M^2 \ominus M^2 = 0$.

Aura-t-on plus de chance en additionnant dans l'autre sens, de la droite vers la gauche ? $S_2 = M \oplus (M^2 \oplus (1 \ominus (M \ominus M^2))) = M \oplus (M^2 \oplus (1 \ominus M^2)) = M \oplus (M^2 \ominus M^2) = M$. Ainsi, les valeurs calculées sont 0 et M alors que la valeur correcte est 1. Que peut-on faire pour améliorer la qualité des calculs et obtenir un résultat juste – ou tout du moins précis – en utilisant uniquement des calculs approchés ?

Un « truc » de bon sens pour tenter d'améliorer la précision d'un calcul de la somme de plusieurs nombres

Fig.2 Plusieurs façons d'arrondir



LES NOMBRES CODÉS par un ordinateur sont finis (traits rouges sur la ligne). Les nombres a et b qui se trouvent chacun entre deux de ces « nombres machine » doivent donc être arrondis. On peut le faire de plusieurs façons : vers le bas, vers le haut, au plus près.

consiste à trier ces nombres par ordre croissant de leur valeur absolue, puis à effectuer le calcul en partant du nombre de plus petite valeur absolue. Par exemple, en décimal et avec une précision de 5, supposons que l'on désire additionner les nombres $a = 1{,}000\,1 \times 10^{0}$; $b = 1{,}200\,1 \times 10^{-4}$; $c = 2{,}400\,2 \times 10^{-4}$ et $d = 4{,}500\,0 \times 10^{-5}$. La somme exacte est 1,000 505 03. Si on additionne dans l'ordre dans lequel on vient de les présenter [c'est-à-dire si l'on calcule $((a \oplus b) \oplus c) \oplus d$ on obtient 1,000 4, tandis que si on les additionne dans l'ordre croissant $((d \oplus b) \oplus c) \oplus a$], on obtient 1,000 5 qui est bien le nombre virgule flottante le plus proche de la valeur exacte. Ici, le truc classique a donné un très bon résultat, mais si on l'applique à notre exemple fil rouge, il ne suffit pas, puisqu'on trouve encore 0 au lieu de 1.

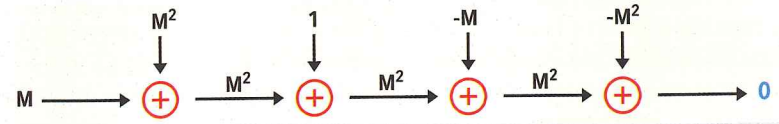
La solution va venir du calcul des erreurs que l'on effectue à chaque étape. Depuis les années 1970, des algorithmes permettent de calculer exactement l'erreur d'une addition virgule flottante. Quand ils sont apparus, beaucoup de ces algorithmes étaient des « recettes de cuisine » : cela marchait sans qu'on sache vraiment pourquoi. Notre travail, comme celui des spécialistes du calcul sur machine, consiste justement à prouver que ces algorithmes fonctionnent, à donner des bornes de fonctionnement rigoureuses. Bref, à nettoyer ces connaissances qui sont souvent empiriques.

Erreur exacte. L'un des algorithmes de sommation les plus répandu a été proposé dans les années 1970 par Donald Knuth, de l'université Stanford, en Californie, prix Turing 1974. Cet algorithme baptisé « TwoSum » fait la somme de deux nombres a et b et donne l'erreur exacte commise lors de cette addition [2]. Cet algorithme n'utilisant que des opérations virgule flottante, son existence montre que l'on peut obtenir, en n'utilisant que des calculs par nature approchés à partir de deux nombres virgule flottante a et b, deux nombres virgule flottante x et y tels que $x = a \oplus b$ et $a + b = x + y$ exactement. >>>

Fig.3 Une addition précise

L'ADDITION $M + M^2 + 1 - M - M^2$ lorsqu'on réalise les opérations de gauche à droite avec un ordinateur donne pour résultat 0 lorsque M est très grand. Grâce à l'utilisation judicieuse de l'algorithme TwoSum, proposé par l'Américain Donald Knuth, on parvient toutefois à obtenir le résultat juste, qui est 1. C'est la méthode la plus rapide.

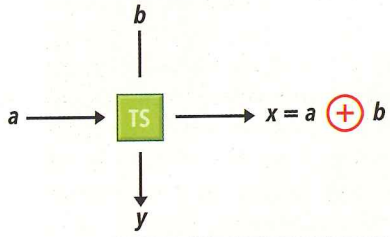
1. ADDITION MACHINE



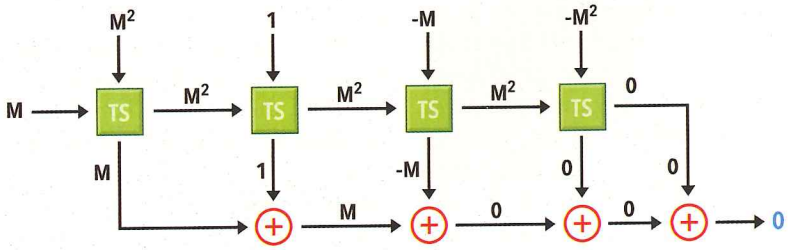
L'addition machine, notée ⊕, fait un arrondi, de telle sorte que, si M est très grand, $M \oplus M^2 = M^2$ par exemple. Le résultat des quatre additions machine successives nécessaires ici donne 0.

2. ALGORITHME TWOSUM

L'algorithme TwoSum (en vert) additionne deux nombres a et b (en entrée) et produit en sortie deux nombres x et y «sans perte de précision». x est le résultat de l'addition machine de a et b et $x + y = a + b$.

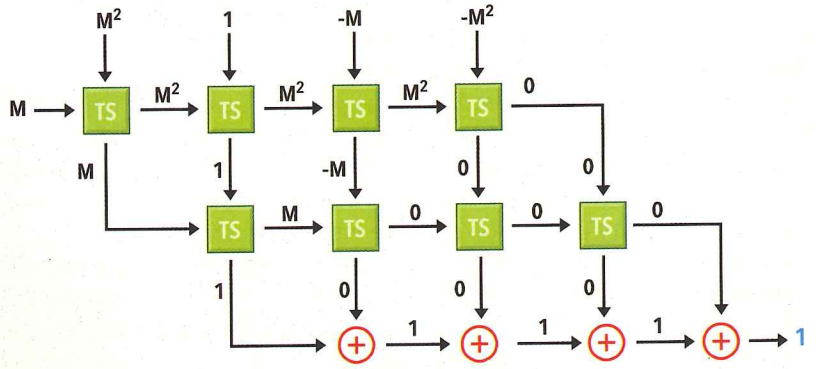


3. ADDITION AVEC TWOSUM.



Si on utilise TwoSum pour réaliser chaque addition (ligne du haut), et que l'on additionne parallèlement entre eux les écarts (lignes du bas) pour corriger l'erreur, on obtient encore 0, comme dans l'addition machine.

4. ADDITION AVEC DEUX FOIS TWOSUM



Pour mieux corriger encore les erreurs d'arrondi, on utilise TwoSum pour l'addition des écarts (ligne du milieu) issus de la première série d'additions. On additionne ensuite simplement les nouveaux écarts obtenus, et l'on somme les résultats des trois lignes : on obtient ainsi le bon résultat.

Des ordinateurs capables de calculer plus juste

››› y est l'erreur commise lors de l'addition de a et b en machine.

Être capable d'obtenir l'erreur commise lorsqu'on effectue une opération va permettre de compenser, du moins partiellement, cette erreur dans la suite du calcul. C'est ce que nous allons voir dans le cas simple de notre exemple récurrent. Mais présentons d'abord l'algorithme TwoSum (a, b) conçu pour faire la somme de a et de b. Il consiste à calculer successivement les valeurs suivantes : $x = a \oplus b$; $b' = x \ominus a$; $a' = x \ominus b'$; $e_b = b \ominus b'$; $e_a = a \ominus a'$; $y = e_a \oplus e_b$.

À la fin de l'exécution de cet algorithme, on conserve les deux valeurs x et y. On a de manière évidente $x = a \oplus b$ et on peut montrer que $y = a + b - x$. Autrement dit, y est l'erreur commise lors du calcul de $a + b$.

Si cet algorithme permet de calculer une somme sans perte de précision, il ne résout pas encore notre problème du calcul de $S = M + M^2 + 1 - M - M^2$. Une première idée consiste à compenser partiellement les erreurs d'arrondis avec un algorithme combinant plusieurs fois TwoSum [fig. 3].

Algorithmes de compensation. De manière générale, pour sommer les nombres $a_1, \ldots a_n$, on effectue d'abord TwoSum(a_1, a_2) qui donnera un résultat $s_2 = a_1 \oplus a_2$ et une erreur e_2, tels que $s_2 + e_2 = a_1 + a_2$, puis TwoSum(s_2, a_3), qui donnera deux valeurs s_3 et e_3, puis TwoSum (s_3, a_4), et ainsi de suite. À la fin de cette étape, on a obtenu un nombre s_n (qui n'est autre que ce qu'on obtiendrait en calculant simplement « $((\ldots(a_1 \oplus a_2) \oplus a_3)\ldots) \oplus a_n$), et des « termes d'erreurs » $e_2, e_3, \ldots, e_n$. On calcule une valeur approchée de la somme des termes d'erreur par la « méthode naïve » c'est-à-dire que l'on calcule $e = ((\ldots(e_2 \oplus e_3) \oplus e_4)\ldots) \oplus e_n$. Le résultat final est $s_n + e$.

Malheureusement, dans notre exemple fil rouge, le terme d'erreur final e est sans intérêt : c'est zéro ! Mais cet algorithme aurait suffi dans le cas plus simple du calcul de $10^{50} + 1 - 10^{50}$ que nous avions présenté précédemment.

L'idée naturelle est alors de compenser plus. Au lieu d'un étage de compensation, on peut en mettre un deuxième. Dans notre exemple, les deux étages suffisent à obtenir le bon résultat, c'est-à-dire 1. Pour des exemples où l'on veut sommer plus de nombres, on peut appliquer la même technique en ajoutant des lignes de compensation. Siegfried Rump, de l'université de Hambourg, en Allemagne, avec Takeshi Ogita et Shin'Ichi Oishi, de l'université Waseda à Tokyo, au Japon, ont montré en 2005 que plus on augmente le nombre de lignes, plus l'algorithme est précis [3].

Ces techniques peuvent sembler coûteuses, car le nombre de calculs augmente à mesure que l'on augmente le nombre de lignes. Toutefois, les calculs virgule flottante sont très rapides et ces algorithmes ne nécessitent aucune comparaison entre les nombres. Ce n'est pas le cas d'un tri (qui pourrait être un précalcul raisonnable, par exemple quand on veut effectuer une opération dans l'ordre croissant des valeurs des nombres) qui nécessite de nombreuses comparaisons et est donc sur la plupart des machines bien plus lent que ce que nous venons de présenter. Nous avons même montré en 2012 que pour la somme de n nombres virgule flottante, l'algorithme TwoSum est le plus efficace possible en termes du nombre d'opérations à effectuer et de vitesse de calcul [4].

De nombreux chercheurs ont utilisé l'algorithme TwoSum ou ses variantes pour améliorer la précision du calcul de la somme d'un grand nombre de valeurs. De telles sommes interviennent dans de multiples problèmes issus de la physique ou des mathématiques (comme l'intégration). De même que l'on est capable, grâce à l'algorithme TwoSum, de connaître l'erreur commise lors du calcul de la somme de deux nombres, on peut connaître l'erreur commise lors du calcul du produit de deux nombres en arithmétique virgule flottante. On construit alors des « algorithmes compensés » qui rattrapent, au moins partiellement, les erreurs commises dans des calculs très variés : produit scalaire, multiplication de matrices, évaluation de polynômes.

Précision améliorée. Au final, c'est un vaste pan du calcul numérique qui est amélioré par ces algorithmes. Bien entendu, même si ces améliorations sont relativement « rapides », elles ont tout de même un coût en temps de calcul, de sorte qu'on les réserve pour les parties critiques d'un programme. Elles sont surtout utiles pour des applications critiques – quand la sécurité est en jeu –, ou quand on soupçonne que le calcul va être peu précis, tel notre exemple où les nombres à additionner ont des valeurs très différentes. Ces briques de calcul au comportement prévisible commencent aussi à être utilisées depuis quelques années dans les logiciels de calcul numérique de sorte qu'on ne devrait plus obtenir des résultats fantaisistes. La haute précision se démocratise. ■

[1] C.-P. Jeannerod *et al.*, *Error Bounds on Complex Floating-Point Multiplication with an FMA*, preprint, 2013 ; S. Graillat et V. Ménissier-Morain, *Information and Computation, 216*, 57, 2012.
[2] D. Knuth, *The Art Computer Programming, Vol. 2*, Seminumerical Algorithms, 3e édition, 1998.
[3] T. Ogita *et al.*, *SIAM Journal of Scientific Computing, 26*, 1955, 2005.
[4] P. Kornerup *et al.*, *IEEE Transac. on Comput., 61*, 289, 2012.

Pour en savoir plus

› Jean-Michel Muller, Arithmétique virgule flottante, École de printemps « précision et reproductibilité en calcul numérique », mars 2013, http://bit.ly/Ecole_PRCN_Muller.

› Valérie Ménissier-Morain, Arithmétique réelle, Séminaire SPIRAL, 2007, www-spiral.lip6.fr/~vmm/expose.pdf

› Vincent Lefèvre et Paul Zimmermann, Arithmétique flottante, Inria, Rapport de recherche n° 5105, 2004, http://bit.ly/Lefevre_Zimmermann.

› J.-C. Bajard et J.-M. Muller (dir.), *Calcul et arithmetique des ordinateurs*, Hermes, 2004.