Comment soustraire des quantités voisines sans perdre de l'information ?

ORDINATEURS EN QUÊTE D'ARITHMÉTIQUE

JEAN-MICHEL MÜLLER

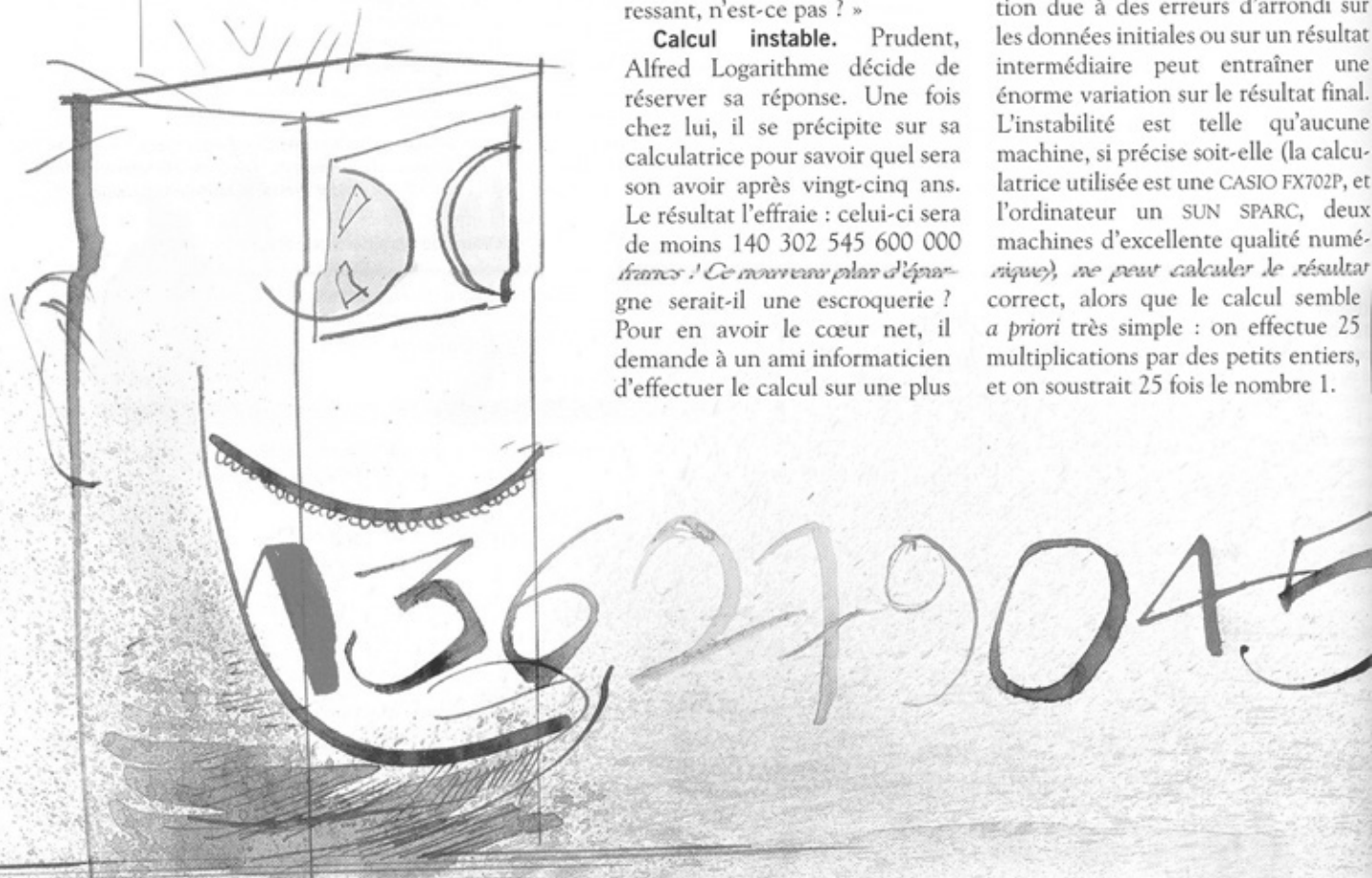
En raison d'erreurs d'arrondi, un puissant ordinateur est susceptible de donner un résultat complètement faux pour des calculs en « virgule flottante », dont l'usage est courant dans l'industrie. Les conséquences pratiques sont potentiellement dévastatrices. De nombreuses recherches s'attachent aujourd'hui à mettre au point des systèmes de représentation des nombres mieux adaptés aux ordinateurs que celui dont nous sommes coutumiers.

JEAN-MICHEL MÜLLER est chargé de recherches CNRS au Laboratoire de l'informatique du parallélisme, à l'Ecole normale supérieure de Lyon.
Il dirige le projet CNRS-INRIA-ENS Lyon Arenaire.

Alfred Logarithme désire faire un placement bancaire à très long terme afin d'assurer l'avenir de sa descendance. Il va se renseigner auprès de la Société chaotique de banque sur son nouveau plan d'épargne. Le banquier lui tient ce discours : « Vous faites un apport initial de (e – 1) francs (e = 2,718 281 828 459 045… est la base des logarithmes naturels). La première année, vous êtes perdant, on multiplie votre capital par 1, et l'on y prélève un franc pour frais de gestion. La deuxième année, c'est mieux, on multiplie votre capital par 2, et l'on prélève toujours un franc pour frais de gestion. La troisième année, on multiplie votre capital par 3 et l'on prélève un franc, et ainsi de suite : la $n^{ième}$ année, on multiplie votre capital par n, et l'on prélève un franc. Au bout de 25 ans, vous pouvez retirer votre argent. Intéressant, n'est-ce pas ? »

Calcul instable. Prudent, Alfred Logarithme décide de réserver sa réponse. Une fois chez lui, il se précipite sur sa calculatrice pour savoir quel sera son avoir après vingt-cinq ans. Le résultat l'effraie : celui-ci sera de moins 140 302 545 600 000 *francs ! Ce nouveau plan d'épargne* serait-il une escroquerie ? Pour en avoir le cœur net, il demande à un ami informaticien d'effectuer le calcul sur une plus grosse machine : il obtient alors un résultat de + 4 645 987 753 francs, ce qui constitue un beau pactole. Rasséréné, Alfred Logarithme court à sa banque pour souscrire à ce plan si intéressant…

Quelle n'est pas sa déconvenue, vingt-cinq ans plus tard, lorsqu'il constate que son avoir est d'environ 4 centimes ! Le cas de ce banquier infernal est un exemple typique de calcul instable, où une infime variation due à des erreurs d'arrondi sur les données initiales ou sur un résultat intermédiaire peut entraîner une énorme variation sur le résultat final. L'instabilité est telle qu'aucune machine, si précise soit-elle (la calculatrice utilisée est une CASIO FX702P, et l'ordinateur un SUN SPARC, deux machines d'excellente qualité numérique), *ne peut calculer le résultat* correct, alors que le calcul semble *a priori* très simple : on effectue 25 multiplications par des petits entiers, et on soustrait 25 fois le nombre 1.



...

Il ressort de cette anecdote que les utilisateurs se font bien souvent des idées fausses sur les ordinateurs. Soit ils leur attribuent une fiabilité supérieure à ce qu'elle est en réalité, soit ils pensent que les algorithmes d'opération mis en œuvre par les ordinateurs ne peuvent qu'être des variantes de ceux que nous avons appris à l'école. Je commencerai donc par présenter quelques problèmes liés à la manipulation des nombres, avant de montrer comment on peut mettre au point des algorithmes très différents des algorithmes classiques.

Dans le calcul d'une suite, les erreurs d'arrondi modifient inévitablement certains termes, perturbant la convergence

Dans l'exemple décrit précédemment, on obtenait des résultats visiblement aberrants, ce qui pouvait éveiller la suspicion. Ce n'est pas toujours le cas, et l'on observe parfois en machine une bonne et rapide convergence vers un résultat totalement faux, comme le montre ce deuxième exemple. Considérons en effet la suite (a_n) de nombres, commençant par $a_0 = 2$, $a_1 = -4$, et dont les termes suivants sont définis par la relation : $a_{n+1} = 111 - 1\,130/a_n + 3\,000/a_n a_{n-1}$. La limite de a_n est 6 (on dit qu'une suite converge vers une limite λ, ou possède une limite λ, si ses termes s'approchent de plus en plus de λ de telle sorte que, pour un seuil donné, aussi petit soit-il, tous les termes de la suite se trouvent, à partir d'un certain rang, à une distance de λ inférieure à ce seuil). Pourtant, sur n'importe quelle machine, on observera une convergence apparente rapide de cette suite vers 100. L'origine de ce phénomène est assez simple : on montre que la suite obtenue en modifiant très légèrement l'un quelconque des termes de la suite (a_n) converge vers 100. Or les erreurs d'arrondi modifient inévitablement certains termes, même si la machine est très précise.

Base et mantisse. Quelles sont les causes d'erreur, et dans quelle mesure peut-on y remédier ? Pour répondre à ces questions, il faut comprendre ce qui se passe exactement en machine lors d'opérations arithmétiques et, avant tout, examiner la manière dont les nombres réels sont représentés dans les ordinateurs.

Supposons, par exemple, que l'on veuille représenter le nombre π écrit avec six chiffres significatifs, soit 3,14159. On vérifie rapidement qu'il est tout à fait équivalent de l'écrire sous la forme $3{,}14159 \times 10^0$ ou encore $0{,}314159 \times 10^1$, etc. La représentation en machine dite virgule flottante s'appuie sur cette équivalence. Un système virgule flottante est essentiellement caractérisé par la donnée d'une base B et d'un nombre M de chiffres utilisés pour représenter ce qu'on appelle la mantisse. L'écriture en virgule flottante d'un nombre x est de la forme :
$x = \pm\, x_0, x_1x_2x_3 \ldots x_{M-1}\, B^E$; le nombre $m = \pm\, x_0, x_1x_2x_3 \ldots x_{M-1}$ est la mantisse de x en base B (ce qui signifie que m est égal à $x_0 + x_1/B + x_2/B^2 + x_3/B^3 + \ldots + x_{M-1}/B^{M-1}$). Le nombre E, l'exposant de B, est un entier.

Dans l'exemple de π, les représentations données plus haut correspondent à B = 10 avec M = 6 et E = 0 dans le premier cas, et M = 7 et E = 1 dans le second. Si M vaut 5 au lieu de 6 ou 7, π s'écrit $3{,}1415 \times 10^0$ ou $0{,}3141 \times 10^1$ pour des valeurs de E égales à 0 et 1 respectivement. Ces deux représentations ne sont pas équivalentes, la première étant plus

LES ARRONDIS : UNE SOURCE IMPORTANTE D'ERREURS

L'addition en virgule flottante s'exécute selon plusieurs phases successives.
Supposons par exemple que l'on travaille en base 10, avec des mantisses de six chiffres, et que l'on désire additionner $X = 9{,}87\,654 \times 10^1$ et $Y = 2{,}00\,071 \times 10^0$. Dans un premier temps, une opération d'alignement est nécessaire :
on réécrit Y avec un exposant égal à 1, soit $Y = 0{,}200\,071 \times 10^1$, afin d'obtenir des nombres de même exposant. Ensuite, on additionne les mantisses : le résultat est 10,076 611 dans cet exemple. Enfin, ce nombre est renormalisé (avec un seul chiffre non nul avant la virgule), c'est-à-dire écrit sous la forme $1{,}0\,076\,611 \times 10^2$, puis arrondi pour ne garder que six chiffres de mantisse, ce qui donne $1{,}00\,766 \times 10^2$ comme résultat final. On a donc perdu les deux derniers chiffres du résultat lors de ce calcul. Mais si la perte de précision est négligeable pour une seule opération, elle l'est beaucoup moins en cas de répétition, comme le montre le calcul de la somme suivante : $1 + 1/2 + 1/3 + \ldots + 1/n$, pour n relativement grand. L'addition étant commutative, ces termes peuvent bien sûr être ajoutés dans n'importe quel ordre. La première idée qui vient à l'esprit est de respecter l'ordre naturel, de gauche à droite (de 1 à 1/n).
Il est toutefois préférable de commencer par les petits termes, pour éviter qu'ils ne deviennent négligeables (1/n est par exemple négligeable devant 1 + 1/2 + 1/3). D'où l'idée de calculer la somme dans l'ordre inverse. Il est même possible de réarranger l'ordre des calculs de manière à garder, pour ce problème, tous les chiffres significatifs, en utilisant des méthodes dues à Michèle Pichat, de l'université de Lyon, et à William Kahan, de l'université de Californie, à Berkeley.

... précise que la deuxième : l'introduction de zéro à gauche des mantisses nuit à la précision, puisqu'il réduit d'autant le nombre de chiffres significatifs. On adopte donc généralement une représentation dite écriture normalisée, où le premier chiffre de la mantisse est non nul. Pour zéro, il faut bien sûr adopter une représentation spéciale.

Les machines n'utilisant pas la base 2 se trouvent aux deux extrêmes de la gamme des ordinateurs : les gros et les calculatrices de poche

Reste que les circuits électroniques utilisés pour effectuer des calculs ou mémoriser les données n'ont généralement que deux états stables (correspondant à deux valeurs de tension électrique), que l'on note par convention 0 et 1. Puisqu'on ne mémorise que des 0 et des 1, chacun des chiffres x_i de la mantisse d'un nombre est écrit en base 2, même lorsque B est différent de 2. Par exemple, si B vaut 10, la mantisse de quatre chiffres 3,141 sera représentée par seize bits (de l'anglais Binary digIT, ou chiffre binaire) de la manière suivante : 0011 0001 0100 0001, où les seize bits sont les conversions respectives en binaire de 3, 1, 4 et 1. C'est la représentation BCD (pour *binary coded decimal*).

L'usage de la base 2 s'est généralisé pour plusieurs raisons. D'une part, ce type de représentation conduisait à une plus grande simplicité des circuits et des programmes de calcul. D'autre part, les études menées essentiellement par l'Américain William Cody(1), alors au laboratoire national d'Argonne, et l'Australien Richard Brent(2), de l'université de Canberra, ont montré qu'elle était la plus satisfaisante du point de vue de la précision. Les machines n'utilisant pas la base 2 se trouvent aux deux extrêmes de la gamme des ordinateurs : certains gros calculateurs et les calculatrices de poche.

Absorption et cancellation. Comment s'effectue alors une addition dans un système virgule flottante (voir l'encadré « Les arrondis : une source importante d'erreurs ») ? Il n'est évidemment pas possible d'additionner directement les mantisses de nombres d'exposants différents. D'où l'opération d'alignement, qui consiste à écrire ces nombres avec le même exposant. Après addition des mantisses, le résultat est renormalisé. Il faut enfin arrondir le résultat obtenu après l'étape de renormalisation pour l'écrire sur M chiffres, ce qui peut entraîner une légère erreur.

Ces erreurs d'arrondi se produisent par exemple lors de l'addition de deux nombres X et Y, avec X très grand en valeur absolue devant Y. A la suite de l'arrondi final, les chiffres les plus à droite de Y sont perdus (voir l'encadré « Les arrondis : une source importante d'erreur »). Il s'agit d'un phénomène dit d'absorption. Ces erreurs se produisent également lors de la multiplication de deux nombres ayant M chiffres de mantisse ; alors que le résultat exact devrait s'écrire avec 2M chiffres, le résultat fourni par l'ordinateur n'en comporte que M : les M chiffres les plus à droite sont perdus. Si cette perte est négligeable pour une seule opération, elle peut ne plus l'être du tout en cas de répétition.

L'autre problème lié à l'arithmétique virgule flottante est appelé cancellation : il survient lors de la soustraction de deux quantités voisines. Notons au passage que le mot de cancellation, qui signifie annulation en anglais, est un vieux mot français que l'on trouve en bonne place dans le Littré. C'est pourquoi je ne cherche pas à le traduire.

Un exemple simple permet de comprendre ce dont il s'agit.

Supposons que l'on travaille en base 10, avec des mantisses de 4 chiffres, et que l'on effectue la soustraction C = A – B, où A et B sont deux nombres virgule flottante : A = $4,205 \times 10^0$ et B = $4,199 \times 10^0$. Le résultat obtenu est : C = $6,000 \times 10^{-3}$. Sauf si A et B sont des valeurs exactes (c'est-à-dire des données initiales ou des résultats de calcul effectués sans erreur), les trois derniers zéro de la mantisse de ce résultat sont artificiels, et il n'est même pas certain que le 6 soit un chiffre significatif. En effet, qui sait si le nombre machine A ne représente pas le réel 4, 205 486, et B le réel 4, 198 787 ? C devrait alors valoir $6,699 \times 10^{-3}$. De même, si A représente 4, 204 786 et B 4, 199 432, le résultat réel est $5,354 \times 10^{-3}$.

Erreur amplifiée. Le phénomène de cancellation est la principale source d'erreur numérique en calcul scientifique. Il agit en fait comme un amplificateur de l'erreur numérique préexistante : il ne crée pas par lui-même de l'erreur. Dans la plupart des calculs instables, des erreurs numériques se créent par cumul d'erreurs d'arrondi, puis sont amplifiées par cancellation. Typiquement, si l'on calcule $(2^{60} + 1) - 2^{60}$ en respectant l'ordre indiqué par les parenthèses, la plupart des ordinateurs ou calculatrices de poche donneront 0 comme résultat au lieu de 1 : l'addition $2^{60} + 1$ produit une erreur d'arrondi, et la soustraction de 2^{60} au résultat produit une cancellation.

A défaut d'éviter les erreurs, comment les estimer ? Les exemples précédents montrent à quel point le résultat d'un calcul peut s'éloigner du résultat exact à cause des erreurs d'arrondi. L'emploi d'une machine plus précise et le respect de certaines règles de programmation limitent éventuellement certains problèmes mais ne les éliminent pas totalement (voir l'encadré « Les arrondis : une source importante d'erreurs »). Deux approches prédominent pour évaluer l'erreur de calcul. La première est déterministe : elle donne un résultat certain. La seconde est probabiliste : le résultat qu'elle fournit est seulement probable. Cette dernière est malgré tout utilisée car elle conduit généralement à des estimations plus fines que les méthodes déterministes.

L'arithmétique d'intervalles relève de la première approche. Elle a été particulièrement étudiée dans les années 1980 par Ulrich Kulisch, de

(1) W.J. Cody, *IEEE Trans. on Computers*, vol. C-22 n° 6, 598, 1973.
(2) R.P. Brent, *IEEE Trans. on Computers*, vol. C-22 n° 6, 601, 1973.
(3) M. Pichat, *Num. Math.*, 19, 400, 1972.
(4) W. Kahan, *Paradoxes in Concepts of Accuracy*, actes du *Joint Seminar on Issues and Directions in Scientific Computation*, U.C. Berkeley, 1989.

l'université de Karlsruhe, et par Willard Miranker, du centre de recherches d'IBM de Yorktown (Etats-Unis)[5]. Le principe est simple. Supposons que l'on cherche à additionner deux nombres machine X et Y (les nombres machine sont notés en majuscules, les nombres réels en minuscules). Le résultat de cette somme calculé par ordinateur est, soit le nombre machine immédiatement inférieur ou égal à la valeur z de la somme réelle — on le note $Z_1 = \nabla(z)$ —, soit le nombre machine immédiatement supérieur ou égal à z – ou $Z_2 = \Delta(z)$. Au lieu de représenter z par Z_1 ou Z_2, pourquoi alors ne pas le représenter par l'intervalle $[Z_1, Z_2]$ auquel il appartient ? L'arithmétique d'intervalle s'appuie précisément sur cette idée, due à Ramon Moore[6] : on représente toutes les variables par des intervalles dont les bornes sont des nombres machine, et l'on effectue les opérations arithmétiques sur ces intervalles ; l'intervalle résultat contient donc toujours le résultat réel exact (voir l'encadré « Des erreurs bien encadrées »). L'erreur de calcul se trouve majorée avec certitude.

Simulation. L'université de Karlsruhe a mis au point des logiciels permettant de faire facilement de l'arithmétique d'intervalles. Celle-ci est utilisée lorsque l'on veut encadrer de manière certaine une solution. En revanche, pour connaître l'ordre de grandeur de l'erreur commise dans une opération en virgule flottante, elle est à proscrire : comme par nature elle majore les erreurs, elle conduit souvent à des estimations pessimistes.

On se tourne alors vers d'autres méthodes, dites de perturbation, qui relèvent d'une approche probabiliste[7,8,9]. L'idée de base est simple et ingénieuse : le meilleur moyen pour tester l'effet, sur le résultat final, des pertes d'information par arrondis n'est-il pas de les simuler soi-même, et de visualiser ainsi leur influence ?

Pour ce faire, il suffit de perturber les calculs en provoquant volontairement, à chaque opération, une erreur aléatoire de l'ordre de grandeur de l'erreur d'arrondi. Pratiquement, on effectue plusieurs fois le même calcul perturbé ; les résultats obtenus sont différents chaque fois compte tenu de la nature aléatoire de l'erreur ; on interprète alors statistiquement les divers résultats obtenus. Cette approche est longtemps restée assez empirique : si les perturbations influaient peu sur le résultat final d'un calcul, on estimait que celui-ci était précis. Un point de vue raisonnable, mais hélas parfois inexact.

C'est seulement depuis 1974 que des modèles statistiques ont été mis au point, afin de mieux tirer parti des calculs perturbés. Signalons par exemple la méthode d'évaluation de l'erreur d'arrondi CESTAC (Contrôle et estimation stochastique des arrondis dans les calculs) de Michel La Porte et de Jean Vignes, respectivement à l'université Pierre-et-Marie-Curie et à l'Institut français du pétrole (IFP). C'est une des seules qui, à ce jour, soit réellement opérationnelle.

Mais plutôt que de se satisfaire des machines existantes et des algorithmes d'opération usuels, pourquoi ne pas imaginer des systèmes de représentation des nombres permettant de calculer plus vite et plus précisément ?

Pour répondre à cette question, examinons l'algorithme d'addition dans notre système de numération en base 10. Soit à effectuer la somme des deux nombres $x_n \ldots x_1 x_0$ et $y_n \ldots y_1 y_0$, où x_0 et y_0 sont les chiffres des unités, x_1 et y_1 les chiffres des dizaines, etc. : on commence par additionner les chiffres de droite, x_0 et y_0. Si leur somme est inférieure strictement à 10, alors elle deviendra le chiffre de droite du résultat ; sinon, celui-ci sera égal à cette somme moins 10, et on créera une retenue égale à 1. On fait de même avec x_1 et y_1 (en tenant compte de la retenue s'il y en a une), et ainsi de suite jusqu'à n.

La principale particularité de cet algorithme est d'être intrinsèquement séquentiel : la retenue au rang i dépend de celle au rang i – 1, etc. Cela ne gêne pas le calculateur humain : si nous sommes capables de faire plusieurs choses simultanément (un informaticien dirait en parallèle) — on marche bien tout en parlant — nous sommes totalement incapables d'effectuer deux calculs différents exactement en même temps.

> On peut imaginer des systèmes de numération dans lesquels l'addition s'effectuerait de manière totalement parallèle

Nous serions donc incapables de calculer en une seule fois tous les chiffres du résultat de l'addition. Que l'algorithme utilisé empêche lui aussi cette simultanéité n'est alors pas très contraignant. Un circuit d'ordinateur est en revanche capable de faire plusieurs calculs en même temps. L'utilisation sur une machine d'un algorithme séquentiel peut donc être très handicapante. Ne serait-il pas possible d'obtenir tous les chiffres d'une somme simultanément ? Dans notre système usuel d'écriture des nombres la réponse est négative : on peut tout au plus accélérer le calcul des retenues. C'est pourquoi il est intéressant d'imaginer des systèmes de numération dans lesquels l'addition s'effectuerait de manière totalement parallèle.

Représentations multiples. En fait, plusieurs systèmes de ce type existent.

DES ERREURS BIEN ENCADRÉES

A la base de l'arithmétique d'intervalles, l'idée d'encadrer chaque nombre réel (par exemple z) par deux nombres machine, l'un immédiatement inférieur et l'autre immédiatement supérieur à z (notés respectivement Z1 et Z2).

Le nombre z est alors représenté par l'intervalle [Z1, Z2]. Cela conduit à définir des règles opératoires, dont nous indiquons ici les plus simples (Δ désigne l'arrondi vers le haut et ∇ l'arrondi vers le bas) :

$[A, B] + [C, D] = [\nabla(A+C), \Delta(B+D)]$

$[A, B] - [C, D] = [\nabla(A-D), \Delta(B-C)]$

$[A,B] \times [C,D] = [\nabla(\min\{AB,AD,BC,BD\}), \Delta(\max\{AC,AD,BC,BD\})]$.

$[A,B] / [C,D] = [A,B] \times (1/[C,D])$.

Les intervalles obtenus à la fin d'un calcul contiennent toujours les valeurs exactes qu'ils représentent ; on est donc certain de majorer l'erreur de calcul. En effet, supposons que l'on souhaite calculer le rapport $t = 1/\sqrt{1-x}$, les nombres étant écrits en base 10 avec quatre chiffres de mantisse, et pour $x = 48/49 = 0{,}979\,591\,836\ldots$ On vérifie aisément que le résultat exact est 7. En machine, l'ordre des opérations est le suivant : il faut calculer $y = 1 - x$, puis $z = \sqrt{y}$, et enfin $t = 1/z$. L'intervalle de départ est $X = [\nabla x, \Delta x] = [9{,}795 \times 10^{-1}, 9{,}796 \times 10^{-1}]$. Tout calcul fait, les nombres machine inférieurs et supérieurs obtenus sont $6{,}983 \times 10^{0}$ et $7{,}003 \times 10^{0}$. Le résultat est bien compris entre ces deux valeurs, mais l'encadrement reste relativement grossier.

9) E. Moore, *Interval Analysis*, Prentice Hall, Englewood Cliffs, New York, 1963.

9) W. Kulisch et W.L. Miranker, *SIAM Rev.*, Vol. 1, 1986.

7) M. La Porte et J. Vignes, *Numer. Math.*, Vol. 39, 1975.

9) J. Vignes, AFCET *Interfaces*, n° 3, 1987.

9) F. Chatelin, *Analyse statistique de la qualité numérique et arithmétique de la résolution approchée d'équations par calcul sur ordinateur*, Rapport F-133, Centre scientifique IBM France, Paris, avril 1988.

... Nous allons en présenter un, dû à Algirdas Avizienis, de l'université de Californie, à Los Angeles[10]. Rappelons que, dans le système usuel, les nombres en base B (par exemple en base 10) sont écrits avec des chiffres compris entre 0 et B – 1 (entre 0 et 9 pour la base 10). Or en 1961, Avizienis a proposé d'écrire les nombres en base B avec des chiffres pris entre – a et +a, où a est compris (au sens large) entre 1 et B – 1, et où 2a + 1 est supérieur ou égal à B. Par exemple en base 10, et en utilisant des chiffres compris entre – 7 et 7 (a = 7), le nombre 1 723 peut s'écrire $2\bar{3}23$, où les chiffres surmontés d'une barre sont considérés comme négatifs ($\bar{3} = -3$). En effet, $2\bar{3}23$ est bien égal à $2 \times 1\,000 + \bar{3} \times 100 + 2 \times 10 + 3 \times 1$. Toutefois, on vérifie facilement que, dans ce système, 1 723 peut également s'écrire $1\,73\bar{7}$, $2\,\bar{3}37$, $22\bar{7}\bar{7}$, (voir l'encadré « Du système décimal usuel aux systèmes redondants »).

Passer de notre système de numération à ceux d'Avizienis ne pose pas de problème majeur

Certains nombres ont donc plusieurs représentations. C'est pourquoi les systèmes d'Avizienis sont souvent qualifiés de redondants. On les appelle également systèmes à chiffres signés. Ils permettent de représenter tous les nombres entiers dès lors que 2a + 1 est supérieur ou égal à B. Par exemple, si a est supérieur ou égal à 5, on pourra représenter tous les nombres en base 10. Les conversions entre notre système de numération usuel et les systèmes d'Avizienis s'effectuent très facilement.

Parallélisme et redondance. Dans son article de 1961, Avizienis proposait un algorithme permettant d'effectuer des additions de manière totalement parallèle (c'est-à-dire en calculant tous les chiffres d'une somme simultanément), à condition que 2a soit supérieur ou égal à B + 1 et que a soit inférieur ou égal à B – 1. Ces conditions sont un peu plus restrictives que celles décrites précédemment. Cet algorithme tire parti de la redondance du système de numération utilisé pour éviter la propagation de retenues au fur et à mesure des sommes partielles : un circuit d'ordinateur peut ainsi calculer tous les chiffres de la somme en même temps (voir l'encadré « L'algorithme d'Avizienis »). La redondance des systèmes de numération d'Avizienis ne va pas sans inconvénients, mais elle est nécessaire : dans sa thèse, Christophe Mazenc, de l'Ecole normale supérieure de Lyon, a montré que, sous certaines conditions, un système de numération permettant d'effectuer des additions de manière totalement parallèle est obligatoirement redondant[11].

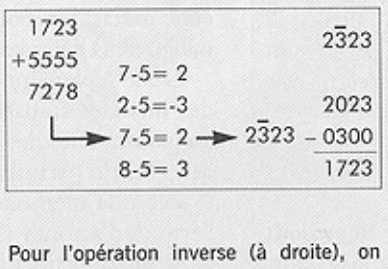
Puisqu'il est possible avec ces systèmes d'effectuer des additions bien plus rapidement, pourquoi ne sont-ils pas plus connus et plus utilisés ? En fait, ils présentent quelques inconvénients. Tout d'abord ils nécessitent beaucoup plus de mémoire et de connexions dans les circuits. Si on veut représenter par exemple tous les nombres entiers compris entre – 2 048 et + 2 047, il suffira de 12 bits en notation usuelle (non redondante) en base 2, tandis qu'il en faudra 24 si on utilise la base 2 et les chiffres – 1, 0 et 1 (2 bits pour représenter chaque chiffre), et 18 si on utilise la base 4 et des chiffres compris entre – 3 et 3. Par ailleurs, si l'addition est plus rapide, les comparaisons entre deux nombres se trouvent ralenties : il est plus difficile de comparer deux nombres qui admettent plusieurs représentations. En outre, la conversion d'un nombre d'un système redondant vers un système non redondant (de même base) prend du temps, celui d'une addition en système usuel (voir l'encadré « Du système décimal usuel aux systèmes redondants »).

Arithmétique en ligne. En conséquence, et bien que ce soit théoriquement possible, la construction d'un ordinateur (c'est-à-dire d'une machine universelle capable de traiter des problèmes très différents) qui soit fondé sur l'utilisation d'un système redondant d'écriture des nombres n'est pas réaliste, du moins en l'état actuel des connaissances et des technologies[12]. En revanche, deux domaines particuliers d'application se dessinent. L'un concerne les architectures spécialisées, c'est-à-dire des machines consacrées à une application particulière (contrôle de processus, traitement numérique du signal, etc.), qui n'ont pas à effectuer les opérations que les systèmes redondants réalisent mal. Le second concerne les opérateurs de type boîte noire : je les appelle ainsi car seul leur concepteur sait qu'ils mettent en œuvre un système de numération spécial. Ces opérateurs, qui sont intégrés dans un ordinateur, reçoivent leurs données et fournissent leur résultat dans un système de numération classique, mais effectuent leur calcul interne dans un système redondant d'écriture des nombres.

Les circuits utilisant l'arithmétique dite en ligne constitue un exemple d'architecture spécialisée. Le pionnier de ce mode de calcul est Milos Ercegovac, professeur à l'université de Californie, à Los Angeles[13]. Les systèmes de calcul en ligne reçoivent leurs données et fournissent les résultats chiffre après chiffre, en commençant par les chiffres les plus significatifs ; il n'y a plus de simultanéité du calcul des chiffres du résultat, mais ceci est compensé par une propriété intéressante : lors de l'enchaînement de plusieurs opérations, un opérateur arithmétique peut travailler sur les premiers chiffres d'un nombre bien avant que l'opérateur précédent ait terminé son calcul. Dans son jargon, l'informaticien appelle ce procédé —

DU SYSTÈME DÉCIMAL USUEL AUX SYSTÈMES REDONDANTS

Pour convertir un nombre représenté en base 10 selon l'écriture usuelle vers l'écriture en « chiffres signés » (à gauche), avec des chiffres pris entre – 5 et 5, on ajoute tout d'abord 555...5 (autant de « 5 » que le nombre à convertir a de chiffres) au nombre que l'on veut convertir, et on enlève 5 à chaque chiffre de la somme (sauf le dernier si cette somme comporte un chiffre de plus que le nombre de départ).



Pour l'opération inverse (à droite), on décompose le nombre à convertir en deux quantités n'ayant que des chiffres positifs : la première est constituée uniquement des chiffres positifs du nombre de départ — en insérant des zéro à la place des chiffres négatifs — et la deuxième est constituée des opposés des chiffres négatifs. Il ne reste plus qu'à soustraire ces deux quantités pour obtenir l'écriture usuelle du nombre de départ.

(10) A. Avizienis, *IRE Transactions on Electronic Computers*, 10, 398, 1961.
(11) C. Mazenc, *Systèmes de représentation des nombres et arithmétique sur machines parallèles*, thèse de doctorat, université Claude Bernard de Lyon et Ecole normale supérieure de Lyon, décembre 1993.
(12) J. Duprat et J.M. Muller, *Technique et Science Informatique*, vol. 10, n° 3, 1991.
(13) M.D. Ercegovac et K. Trivedi, *IEEE Transactions on Computers*, vol. C-26 n° 7, 1977.

très proche du principe du travail à la chaîne — un pipe-line au niveau du chiffre (voir l'encadré « Du calcul "en ligne" au "pipe-line" »). Il est ainsi possible d'effectuer très rapidement des séquences complexes de calculs.

Précision accrue. Les premiers algorithmes en ligne pour l'addition, la multiplication et la division datent de 1977. Pour les autres fonctions, certains ont été mis au point dans les années 1980, d'autres constituent encore un domaine de recherche actif. Au Laboratoire de l'informatique du parallélisme, à l'Ecole normale supérieure de Lyon, nous travaillons actuellement à la conception d'algorithmes et de circuits pour le calcul en ligne des fonctions sinus, cosinus, exponentielle et logarithme. L'arithmétique en ligne peut conduire à une meilleure maîtrise de la précision : dans une chaîne de calculs, l'utilisateur reçoit les chiffres du résultat un par un et peut prolonger le calcul jusqu'à ce qu'il ait obtenu toute la précision désirée. De telles voies ont été explorées en France par Christophe Mazenc (cité plus haut), et Valérie Ménissier, de l'Institut national de recherche en informatique et en automatique (INRIA), à Rocquencourt[11,14]. Valérie Ménissier a par exemple écrit une bibliothèque de programmes de manipulation de nombres réels à précision arbitrairement grande, désormais intégrée dans un langage de programmation (le langage CAML).

Lorsque le volume de calcul est important par rapport à une addition, on utilise de préférence des opérateurs de type boîte noire

Quant aux opérateurs de type boîte noire, ils ne sont efficaces que si le volume de calcul à effectuer est important par rapport à une addition avec propagation de retenue (addition selon l'algorithme usuel), puisque la conversion des données, à l'entrée et à la sortie de l'opérateur, prend le temps d'une telle addition. Aujourd'hui, la plupart des multiplicateurs (c'est-à-dire les circuits qui effectuent des multiplications dans les ordinateurs) utilisent pour leurs calculs internes une représentation appelée *carry-save*, qui est un système redondant d'écriture des nombres en base 2 avec les chiffres 0, 1 et 2. Mais en 1985, Naofumi Takagi, Hiroto Yasuura et Shuzo Yajima, de l'université de Kyoto, ont proposé un nouveau multiplicateur particulièrement performant, au sein duquel les calculs se font en base 2 avec les chiffres – 1, 0 et 1[15]. C'est un multiplicateur de ce type qui a été implanté dans un des coprocesseurs mathématiques construits par la société Cyrix, que l'on peut placer dans un micro-ordinateur pour accélérer les calculs. Depuis, de nombreux opérateurs boîte noire pour la division ont vu le jour.

Pour notre part, nous nous intéressons également de près à la conception d'un tel opérateur pour calculer les fonctions mathématiques de base (sinus, cosinus, exponentielle, logarithme, etc.)[16]. Nous avons mis au point un algorithme adapté à ce genre d'opérateurs. Celui-ci présente la particularité de n'utiliser que des additions (qui s'effectuent très vite si on utilise un système d'Avizienis pour représenter les nombres) et des divisions par des puissances de 2.

Une puissance de 2 est un nombre de la forme $2 \times 2 \times 2 \times \ldots \times 2$: diviser par un tel nombre lorsque l'on travaille en base 2 est aussi simple que diviser par 10, 100 ou 1 000 lorsque l'on travaille en base 10 ; cela se fait par un simple décalage de la virgule. Notre algorithme permet de calculer rapidement, entre autres, les fonctions sinus, cosinus, logarithme, exponentielle, arctangente et racine carrée. Du fait de ses qualités (rapidité, grand nombre de fonctions calculables), il pourrait être utilisé dans les calculettes et les processeurs virgule flottante.

Confiance limitée. Cet article avait pour objet de présenter deux domaines importants de l'arithmétique des ordinateurs, l'évaluation d'erreurs et le calcul rapide grâce

L'ALGORITHME D'AVIZIENIS

Le professeur Algirdas Avizienis est notamment professeur émérite à l'université de Californie, à Los Angeles, Etats-Unis.

(Cliché J. Jamulaitis)

L'algorithme d'Avizienis permet d'effectuer des additions de manière totalement parallèle. Il est fondé sur une représentation particulière des nombres : chaque nombre en base B est écrit avec des chiffres compris (au sens large) entre a et – a, où 2a est supérieur ou égal à B+1, et a inférieur ou égal à B – 1 (à condition que B soit différent de 2 ; l'algorithme n'est donc pas compatible avec la base 2).

Le résultat $s_{n+1}\ldots s_1\ s_0$ de l'addition de deux nombres $x_n\ldots x_1\ x_0$ et $y_n\ldots y_1\ y_0$ en base B s'obtient en calculant en parallèle la somme $w_i + t_i$, pour i de 0 à n+1, où w_i et t_i sont eux-mêmes calculés en parallèle de la manière suivante : pour i de 0 à n

$$t_{i+1} = \begin{cases} -1 \text{ si } x_i + y_i \leq -a \\ +1 \text{ si } x_i + y_i \geq +a \\ 0 \text{ si } -a+1 \leq x_i + y_i \leq a-1 \end{cases}$$

$w_i = x_i + y_i - Bt_{i+1}$

et, par convention : $w_{n+1} = t_0 = 0$

Les termes t_{i+1} jouent un rôle similaire à celui des retenues dans une somme usuelle, à la différence près qu'il n'y a aucune dépendance entre t_i et t_{i+1}, ce qui permet le parallélisme. Il est important de comprendre qu'en utilisant cet algorithme, un circuit d'ordinateur peut calculer tous les chiffres de la somme en même temps. Par exemple, supposons que l'on souhaite additionner les deux nombres écrits en base 10 avec $a = 6 : 1\bar{2}45$ (= 765) et $\bar{2}3\bar{3}2$ (= – 2328) (ou $\bar{2} = -2$).

Le résultat obtenu, soit $1\bar{6}4\bar{3}$ (= – 1 563) est bien égal à la somme de 765 et – 2 328.

En base 2, et en utilisant les chiffres – 1, 0 et 1, il est également possible de concevoir des algorithmes complètement parallèles d'addition qui ne sont pas très différents de l'algorithme présenté plus haut.

i	3	2	1	0
x_i	1	$\bar{2}$	$\bar{4}$	5
y_i	$\bar{2}$	$\bar{3}$	$\bar{3}$	2
x_i+y_i	–1	–5	–7	7
t_{i+1}	0	0	–1	1
w_i	–1	–5	$\bar{3}$	–3
s_i	$\bar{1}$	$\bar{6}$	4	3

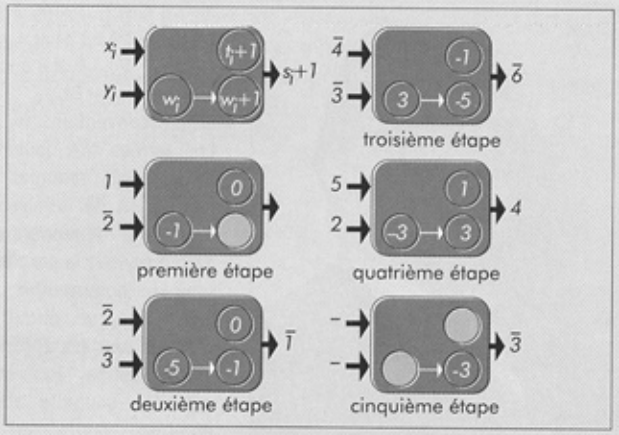
Ces algorithmes permettent d'additionner deux nombres écrits sur n chiffres (pour n quelconque) en un peu plus que le temps nécessaire à un additionneur classique pour ajouter deux nombres de 2 chiffres.

14) Ménissier, Arithmétique exacte », thèse de doctorat, Université Paris VII, décembre 1994.

15) N. Takagi et al., *IEEE Transactions on Computers*, vol. C-34 n° 9, septembre 1985.

16) D. Bajard et al., *IEEE Transactions on Computers*, vol. 43, n° 8, août 1994.

DU CALCUL « EN LIGNE » AU « PIPE-LINE »



Les systèmes redondants de représentation des nombres permettent de mettre en œuvre un mode de calcul particulier dit en ligne, qui est schématisé ici. Réexaminons l'algorithme d'addition d'Avizienis, présenté dans l'encadré « Du système décimal usuel aux systèmes redondants », et supposons que nous recevions les chiffres des données un par un, de la gauche vers la droite (on reçoit tout d'abord x_n et y_n, puis x_{n-1} et y_{n-1}, etc.).

A partir de x_n et y_n, on est capable d'en déduire t_{n+1} et w_n. A l'étape suivante, x_{n-1} et y_{n-1} permettent de calculer t_n et w_{n-1}. On peut alors fournir le premier chiffre du résultat en partant de la gauche, s_n, qui est égal à $w_n + t_n$.

A l'étape suivante, le processus est le même avec x_{n-2} et y_{n-2}, et ainsi de suite : des valeurs x_i et y_i, on déduit celles de t_{i+1} et de w_i, et par conséquent s_{i+1} (selon l'algorithme d'Avizienis). Sur le schéma ci-dessus sont représentées les différentes étapes de l'addition, à partir de l'exemple donné dans l'encadré « Du système décimal usuel aux systèmes redondants », c'est-à-dire l'addition de $1\bar{2}45$ (= 765) et $\bar{2}\bar{3}32$(= – 2328). L'intérêt d'un tel mode de calcul, appelé en ligne, est que les premiers chiffres du résultat deviennent disponibles pour le calcul suivant bien avant que le calcul en cours ne soit terminé :

sn devient disponible pour l'opération suivante dès que x_{n-1} et y_{n-1} ont été reçus, etc.

Si on faisait circuler les chiffres de la droite vers la gauche, on pourrait construire des opérateurs arithmétiques ayant un comportement similaire pour l'addition et la multiplication (et ceci sans avoir besoin d'un système redondant de représentation des nombres), mais la plupart des autres calculs seraient impossibles.

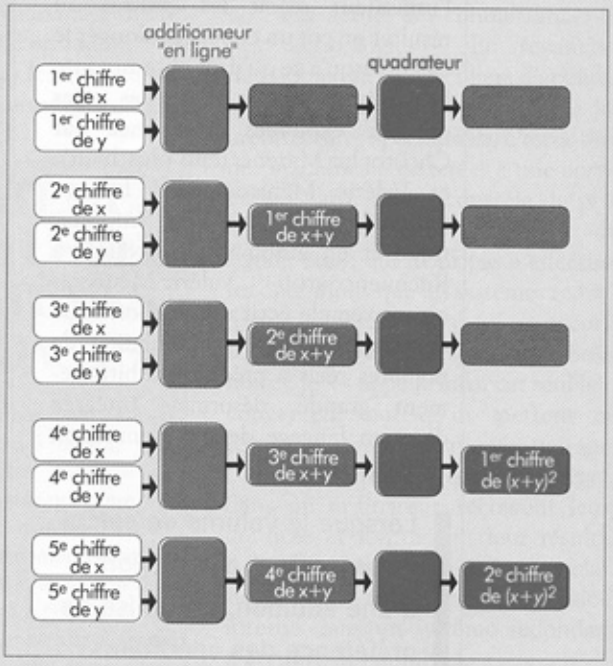
En arithmétique en ligne, on sait construire de tels opérateurs pour presque toutes les fonctions mathématiques usuelles (division, racine carrée, sinus, cosinus, logarithme, exponentielle, etc.). Cet algorithme d'addition permet d'obtenir le $(i-1)^{ème}$ chiffre (en partant de la gauche) de la somme de deux nombres dès que l'on a entré dans l'opérateur les $i^{èmes}$ chiffres de ces nombres. On dit que l'addition est de délai 1 : un opérateur sera de délai d s'il fournit le $(i-d)^{ème}$ chiffre du résultat après réception des ièmes chiffres de ses entrées. On sait faire par exemple des multiplications « en ligne » avec un délai égal à 2.

La figure ci-dessous présente ce qui se passe lorsqu'on calcule $(x+y)^2$ en arithmétique « en ligne » avec un additionneur de délai 1 et un quadrateur (c'est-à-dire un opérateur qui élève au carré son entrée) de délai 2.

Dès réception du $i^{ème}$ chiffre de x et y, on obtient le $(i-1)^{ème}$ chiffre de $x + y$, dont on déduit le $(i-3)^{ème}$ chiffre de $(x+y)^2$. L'addition et l'élévation au carré s'effectuent donc en même temps, et on peut se servir des premiers chiffres de $x + y$ bien avant que le calcul de $x + y$ ne soit terminé.

Cette technique bien connue des informaticiens, et qui rappelle le travail à la chaîne, est appelée « pipe-line ».

En utilisant cette technique, le délai global d'une séquence de calcul sera égal à la somme des délais des opérateurs utilisés.



à l'utilisation d'une arithmétique redondante.

Si le lecteur retient de tout ceci qu'il ne faut jamais faire aveuglément confiance aux résultats fournis par un ordinateur et que l'implantation rapide d'une opération aussi élémentaire que l'addition relève encore du domaine de la recherche, je serai pleinement satisfait.

Barrages et missiles. Les exemples numériques donnés au début de l'article sont volontairement ludiques : ce sont des exemples d'école, mais il ne faut pas pour autant oublier qu'ils mettent en évidence un problème réel. Une longue suite de calculs peut conduire à un résultat numérique très imprécis, et les conséquences d'une telle imprécision peuvent être graves : des calculs virgule flottante sont faits à l'intérieur des missiles, des avions. L'arithmétique virgule flottante est utilisée pour concevoir des barrages, des ponts, des automobiles, etc.

Dans sa thèse de doctorat, Douglas Priest, de l'université de Californie, à Berkeley, cite le cas d'un missile *Patriot* n'ayant pas réussi à intercepter un missile irakien (c'était pendant la guerre du Golfe) à la suite d'une erreur numérique.

Le grand mathématicien Anston Householder disait qu'il avait peur de monter dans un avion depuis qu'il savait que ceux-ci étaient conçus en utilisant l'arithmétique virgule flottante : c'était de sa part une plaisanterie, mais espérons que l'avenir ne lui donnera pas raison.

J.-M.M. ■

Pour en savoir plus

■ D. Goldberg, *What Every Computer Scientist should know about Floating-Point Arithmetic*, ACM Computing Surveys, vol. 23, nº 1, mars 1991.

■ A.R. Omondi, *Computer Arithmetic Systems*, Prentice Hall International series in Computer Science, 1994.

■ M. Daumas et J.M. Muller (coordinateurs), *Qualité des calculs sur ordinateur*, Masson, France, 1997.

■ J.-M. Muller, *Elementary Functions, Algorithms and Implementation*, Birkhauser Boston, 1997.