Arithmétique réelle

Introduction à l'arithmétique flottante
La précision des calculs: analyse et améliorations

Valérie Ménissier-Morain

Université Pierre et Marie Curie - Paris 6
LIP6 - Département CALSCI

Séminaire SPIRAL - 1er juin 2007

Plan

Représentation flottante

Un nombre x est représenté en virgule flottante en base B par :

- son signe s_x (0 pour x positif, 1 pour x négatif)
- sa mantisse m_x de $n+1$ chiffres
- son exposant e_x, un entier de k chiffres compris entre e_{min} et e_{max}

tels que

$$x = (-1)^{s_x} \times m_x \times 2^{e_x}$$

avec $m_x = x_0.\, x_1\, x_2\, \ldots\, x_n$ où $x_i \in \{0,\, 1,\, \ldots,\, B-1\}$.

Pour obtenir la meilleur précision à longueur de mantisse n fixée, la mantisse est normalisée, c'est-à-dire que $x_0 \neq 0$, $m_x \in]1, B[$.

Il faut donc un codage spécial pour 0.

Un principe unique mais la cacophonie. . .

Le résultat d'une opération diffère selon le langage, le compilateur et l'architecture de la machine

machine	B	n	e_{min}	e_{max}
Cray 1	2	48	-8192	8191
	2	96	-8192	8191
DEC VAX	2	53	-1023	1023
	2	56	-127	127
HP 28 et 48G	10	12	-499	499
IBM 3090	16	6	-64	63
	16	14	-64	63
	16	28	-64	63

Impossible d'écrire du code numérique portable

. . . et les propriétés les plus élémentaires de l'arithmétique réelle souvent mises à mal

Sur Cray par exemple,

$$x + y \neq y + x$$

et

$$0.5 \times x \neq x/2.0$$

Impossible d'écrire du code numériquement correct simple ou de raisonner sur les propriétés mathématiques de l'arithmétique flottante

Révolution en 1985 : la norme IEEE-754

Après de nombreuses années de travail, une norme fixe la représentation des données et le comportement des opérations de base en virgule flottante.
Cette norme fixe :

- les formats des données
- les valeurs spéciales
- les modes d'arrondi
- la précision des opérations de base
- les règles de conversion

En fait, il y a deux normes :

754 *ANSI/IEEE Standard for Binary Floating-Point Arithmetic* en 1985

854 *ANSI/IEEE Standard for Radix-Independent Floating-Point Arithmetic* en 1987 (où $B = 2$ ou 10)

Objectifs de la norme IEEE-754

- permettre de faire des programmes portables
- rendre les programmes déterministes d'une machine à une autre
- conserver des propriétés mathématiques
- permettre/imposer l'arrondi correct
- permettre/imposer des conversions fiables
- faciliter la construction de preuves
- faciliter la gestion des exceptions (fonctions partielles)
- faciliter les comparaisons (unicité de la représentation, sauf pour 0)
- permettre un support pour l'arithmétique d'intervalle

Norme IEEE-754 : Formats de base

$B = 2$

format	nombre de bits			
	total	signe	mantisse	exposant
simple précision	32	1	(1 implicite +) 23 (fraction)	8
double précision	64	1	(1 implicite +) 52 (fraction)	11

La mantisse (normalisée) m_x du nombre flottant x est représentée par $n+1$ bits :

$$m_x = 1.\underbrace{x_1\ x_2\ x_3\ \ldots\ x_{n-1}\ x_n}_{f_x}$$

où les x_i sont des bits.

La partie fractionnaire de m_x est appelée fraction (de n bits) : f_x.
On a alors : $m_x = 1 + f_x$ et $1 \leq m_x < 2$.

Norme IEEE-754 : exposant

L'exposant e_x est un entier signé de k bits tel que $e_{min} \leq e_x \leq e_{max}$

Différentes représentations sont possibles :

- complément à 2,
- signe et magnitude,
- biaisée.

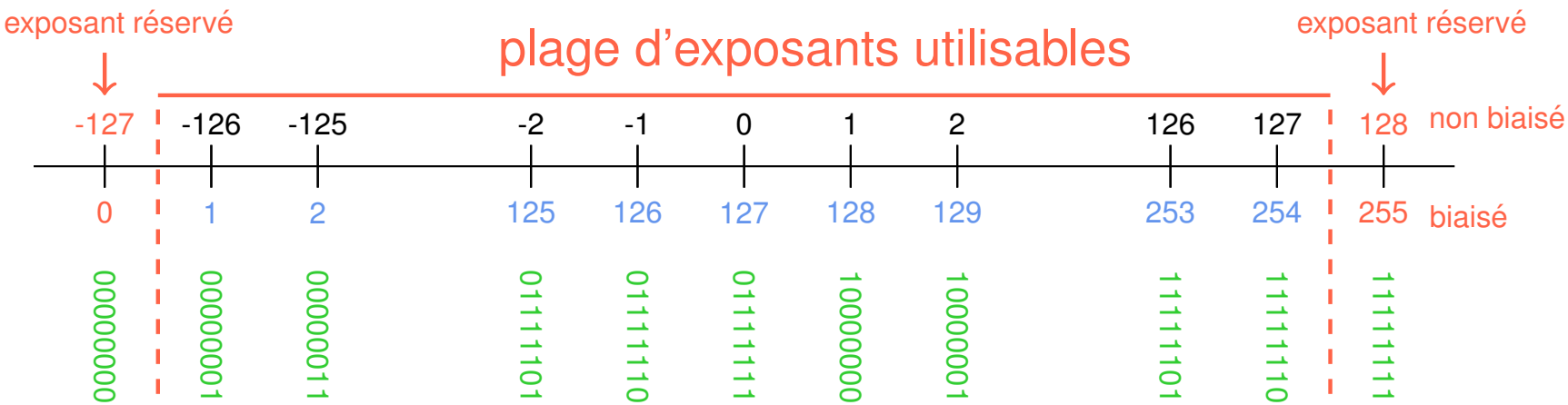
Norme IEEE-754 : choix de la représentation biaisée stockée avant la mantisse car cela permet de faire les comparaisons entre flottants dans l'ordre lexicographique (en laissant le signe s_x de côté) et de représenter le nombre 0 avec $e_x = f_x = 0$.

L'exposant stocké physiquement est l'exposant biaisé e_b tel que $eb_x = e_x + b$ où b est le biais.

Norme IEEE-754 : exposants réservés

Les exposants non biaisés $e_{min} - 1$ et $e_{max} + 1$ (respectivement 0 et $2^k - 1$ en biaisé) sont réservés pour zéro, les dénormalisés et les valeurs spéciales.

format	taille	biais	non-biaisée		biaisée	
	k	b	e_{min}	e_{max}	eb_{min}	eb_{max}
SP	8	$127\ (= 2^{8-1} - 1)$	-126	127	1	254
DP	11	$1023\ (= 2^{11-1} - 1)$	-1022	1023	1	2046



Norme IEEE-754 : représentation de zéro

$e_0 = f_0 = 0$, $s_0 = ?$

Deux représentations différentes : -0 et +0.

Ce qui est cohérent avec le fait qu'il y ait deux infinis distincts. On a alors : $\frac{1}{+0} = +\infty$ et $\frac{1}{-0} = -\infty$.

La norme impose que le test $-0 = +0$ retourne la valeur vrai.

En simple précision, la représentation machine de +0 et -0 est donc :

x	s_x	eb_x	m_x
-0	1	00000000	00000000000000000000000
+0	0	00000000	00000000000000000000000

Norme IEEE-754 : valeurs spéciales

- Les infinis, $-\infty$ et $+\infty$: $e_x = e_{max} + 1$ et $f_x = 0$.
- Not a Number : NaN (exception, fonction partielle)

  Le résultat d'une opération invalide telle que $0/0$, $\sqrt{-1}$ ou $0 \times +\infty$: $e_x = e_{max} + 1$ et $f_x \neq 0$.

  Les NaN se propagent dans les calculs.

Norme IEEE-754 : valeurs spéciales

- Les infinis, $-\infty$ et $+\infty$: $e_x = e_{max} + 1$ et $f_x = 0$.
- Not a Number : NaN (exception, fonction partielle)

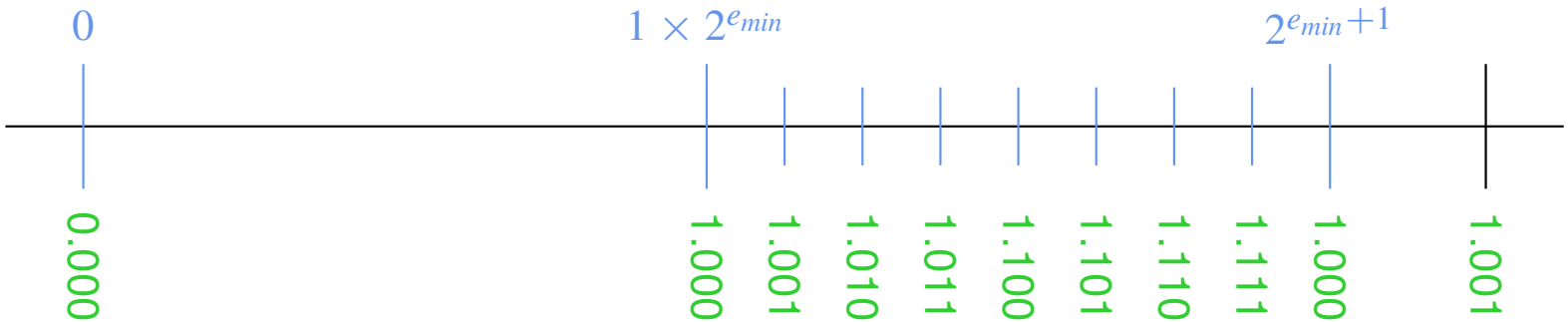
  Le résultat d'une opération invalide telle que $0/0$, $\sqrt{-1}$ ou $0 \times +\infty$: $e_x = e_{max} + 1$ et $f_x \neq 0$.

  Les NaN se propagent dans les calculs.

En simple précision, on a donc :

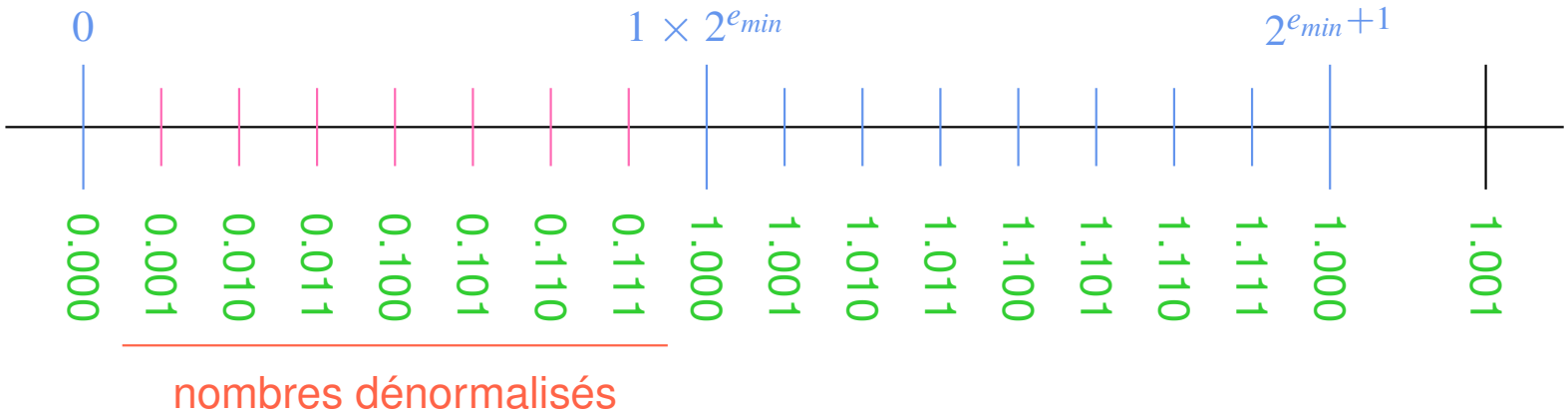
x	s_x	eb_x	m_x
$-\infty$	1	11111111	00000000000000000000000
$+\infty$	0	11111111	00000000000000000000000
NaN	0	11111111	00001001001111000000100 (par exemple)

Norme IEEE-754 : autour de zéro



Le plus petit nombre positif normalisé x est tel que $e_x = e_{min}$, $f_x = 0$, donc $x = 2^{e_{min}}$.
Le « 1 implicite » dans la mantisse implique qu'il n'y a pas de nombre représentable entre $0 \times 2^{e_{min}}$ et $1 \times 2^{e_{min}}$ alors qu'il y en a 2^n entre $1 \times 2^{e_{min}}$ et $2 \times 2^{e_{min}}$.

Norme IEEE-754 : autour de zéro, les nombres dénormalisés



Le plus petit nombre positif normalisé x est tel que $e_x = e_{min}$, $f_x = 0$, donc $x = 2^{e_{min}}$.

Le « 1 implicite » dans la mantisse implique qu'il n'y a pas de nombre représentable entre $0 \times 2^{e_{min}}$ et $1 \times 2^{e_{min}}$ alors qu'il y en a 2^n entre $1 \times 2^{e_{min}}$ et $2 \times 2^{e_{min}}$.

L'objectif des nombres dénormalisés est d'uniformiser la répartition des nombres représentables autour de 0.

On autorise près de zéro des nombres de la forme $(-1)^{s_x} \times 0.f_x \times 2^{e_{min}}$: $eb_x = 0$, m_x ne suit pas la convention du « 1 implicite ».

Norme IEEE-754

nombres normalisés et exposants réservés en bref

Exposant non biaisé	Mantisse	Commentaire
$e_{min} \leq e_x \leq e_{max}$		nombre normalisé, convention du « 1 implicite »
$e_x = e_{min} - 1$	$m_x = 0$	zéro
	$m_x \neq 0$	nombre dénormalisé, proche de zéro, pas de convention du « 1 implicite »
$e_x = e_{max} + 1$	$m_x = 0$	infini $(-1)^{s_x} \infty$
	$m_x \neq 0$	NaN, résultat d'une opération invalide, se propage dans le calcul, NaN$\neq$NaN

L'utilisation d'exposants réservés conduit à des calculs plus coûteux.

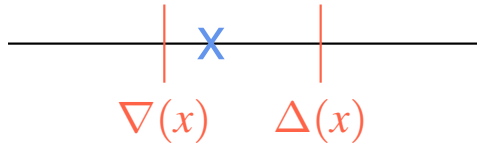
Norme IEEE-754 : nécessité d'arrondir

Si x et y sont deux nombres exactement représentables en machine, alors le résultat d'une opération $res = x \odot y$ n'est, en général, pas représentable en machine.

Par exemple, en base $B = 10$, le nombre $1/3$ n'est pas représentable avec un nombre fini de chiffres.

Il faut arrondir le résultat, c'est-à-dire renvoyer un des nombres représentables voisins.

Norme IEEE-754 : modes d'arrondis



La norme propose 4 modes d'arrondi :

- arrondi vers $+\infty$ (ou par excès), noté $\Delta(x)$: retourne le plus petit nombre machine supérieur ou égal au résultat exact x
- arrondi vers $-\infty$ (ou par défaut), noté $\nabla(x)$: retourne le plus grand nombre machine inférieur ou égal au résultat exact x
- arrondi vers 0, noté $\mathcal{Z}(x)$: retourne $\Delta(x)$ pour les nombres négatifs et $\nabla(x)$ pour les positifs
- arrondi au plus près, noté $\circ(x)$: retourne le nombre machine le plus proche du résultat exact x (pour le milieu de deux nombres machine consécutifs on choisit celui dont la mantisse se termine par un 0, on parle d'arrondi pair)

Les trois premiers modes d'arrondis sont dits dirigés.

Norme IEEE-754 : propriété de l'arrondi correct

Soient x et y sont deux nombres exactement représentables en machine, $\odot$ une des opérations rationnelles $+$, $-$, $\times$, $/$ et $\diamond$ le mode d'arrondi choisi parmi les 4 modes IEEE.

La norme IEEE exige que le résultat d'une opération $x \odot y$ soit égal à $\diamond(x \odot_{exact} y)$. Le résultat doit être le même que si on effectuait le calcul en précision infinie puis on arrondissait ce résultat.

Idem pour la racine carrée.

C'est la propriété de l'arrondi correct.

La norme IEEE décrit un algorithme pour l'addition, a soustraction, la multiplication, la division et la racine carrée et exige que ses implémentations produisent le même résultat que cet algorithme.

Norme IEEE-754 : comparaisons

La norme impose que l'opération de comparaison soit exacte et ne produise pas de dépassement de capacité.

Les comparaisons spécifiées dans la norme sont :

- égalité
- supérieur
- inférieur

Le signe de zéro n'est pas pris en compte.

Dans le cas de comparaisons impliquant un NaN, la comparaison retourne faux.
Plus précisément dans le cas de l'égalité : si $x = \text{NaN}$ alors $x = x$ retourne faux et $x \neq x$ retourne vrai (l'égalité n'est pas réflexive mais c'est un moyen d'identifier un NaN)

Norme IEEE-754 : les drapeaux

Aucun calcul ne doit entraver le bon fonctionnement de la machine. Un mécanisme de 5 drapeaux permet d'informer le système sur le comportement des opérations :

INVALID operation le résultat par défaut est NaN

DIVIDE by ZÉRO le résultat est $\pm\infty$

OVERFLOW dépassement de capacité vers ∞ : le résultat est soit $\pm\infty$ soit le plus grand nombre représentable (en valeur absolue) suivant le signe du résultat exact et du mode d'arrondi

UNDERFLOW dépassement de capacité vers 0 : le résultat est soit ± 0 soit un dénormalisé

INEXACT résultat inexact : levé lorsque que le résultat d'une opération n'est pas exact (presque toujours ignoré)

Ces drapeaux, une fois levés, le restent pendant tout le calcul jusqu'à une remise à zéro volontaire (sticky flags). Ils peuvent être lus et écrits par l'utilisateur.

Norme IEEE-754

Ce que je ne vais pas vous raconter en détails

La norme spécifie les conversions binaire-décimal avec la règle de la double conversion.

Il existe des formats étendus définis moins précisément dans la norme et pas toujours implémentés.

Les calculs se font avec des bits supplémentaires appelés bits de garde.
La norme est en cours de révision depuis plusieurs années, notamment pour étendre la propriété de l'arrondi exact à toutes les fonctions élémentaires (déjà implémenté dans crlibm, voir plus loin) : IEEE-754r. Cela a nécessité un gros effort de tabulation pour déterminer des bornes pour le dilemme du fabricant de table.

Un résumé de la norme IEEE-754 en chiffres

	exposant	fraction	valeur
normalisés	$e_{min} \leq e \leq e_{max}$	$f \geq 0$	$\pm(1.f) \times 2^e$
dénormalisés	$e = e_{min} - 1$	$f > 0$	$\pm(0.f) \times 2^{e_{min}}$
zéro (signé)	$e = e_{min} 1$	$f = 0$	± 0
infinis	$e = e_{max} + 1$	$f = 0$	$\pm\infty$
Not a Number	$e = e_{max} + 1$	$f > 0$	NaN

format	# bits total	k	n	e_{min}	e_{max}	b
simple précision	32	8	23	-126	127	127
double précision	64	11	52	-1022	1023	1023

valeur	simple précision	double précision
grand normalisé > 0	$3.40282347 \times 10^{38}$	$1.7976931348623157 \times 10^{308}$
petit normalisé > 0	$01.17549435 \times 10^{-38}$	$2.2250738585072014 \times 10^{-308}$
dénormalisé > 0	$1.17549421 \times 10^{-38}$	$2.2250738585072009 \times 10^{-308}$
dénormalisé > 0	$1.40129846 \times 10^{-45}$	$4.9406564584124654 \times 10^{-324}$

Plan

Précision des calculs

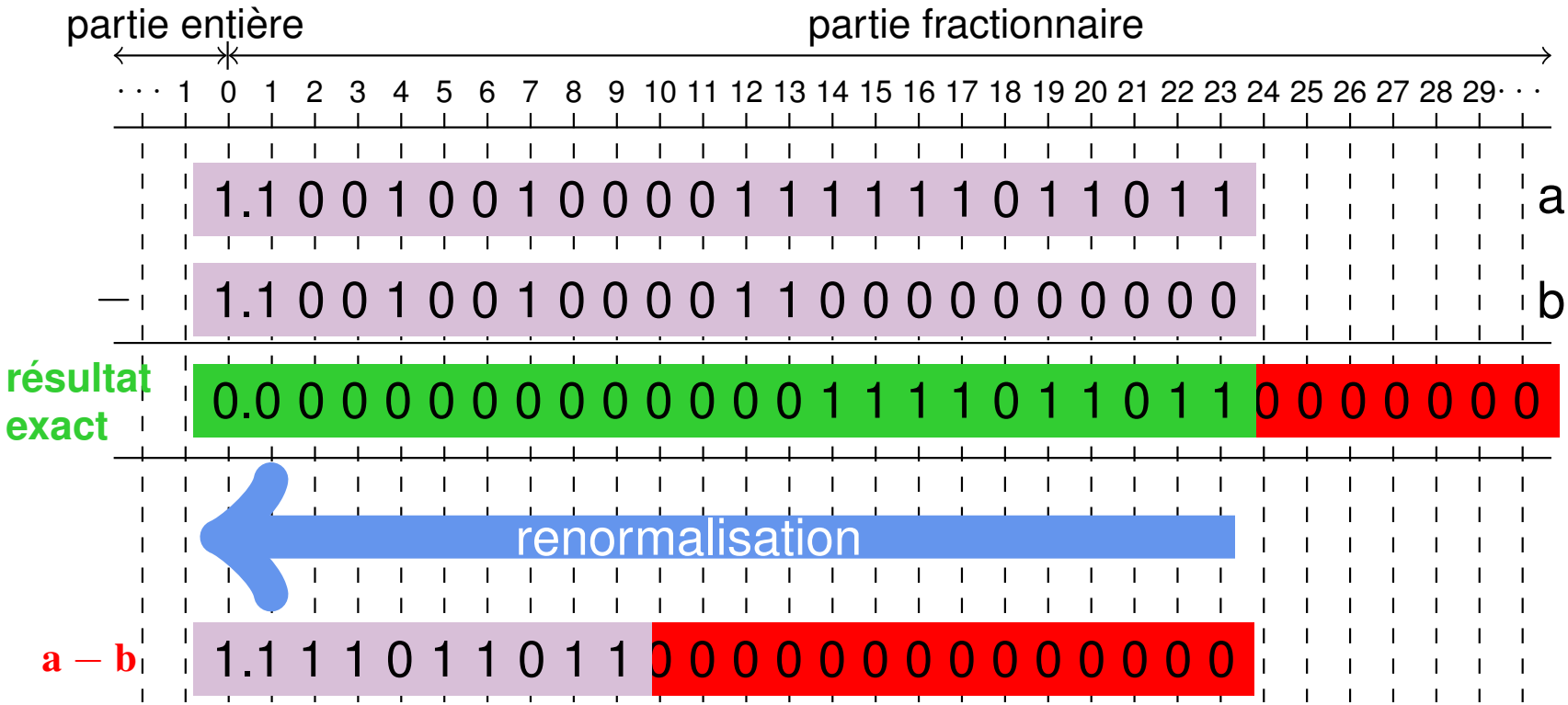
À chaque arrondi, on perd a priori un peu de précision, on parle d'erreur d'arrondi.

Même si une opération isolée retourne le meilleur résultat possible (l'arrondi du résultat exact), une suite de calculs peut conduire à d'importantes erreurs du fait du cumul des erreurs d'arrondi.

Les deux sources principales d'erreur d'arrondis au cours des calculs sont la *cancellation* et l'absorption. Mais ces erreurs n'interviennent qu'après des mesures physiques approximatives, une modélisation physique hasardeuse et un algorithme de résolution plus ou moins approchée et correctement implémenté.

Phénomène de *cancellation* (ou élimination)

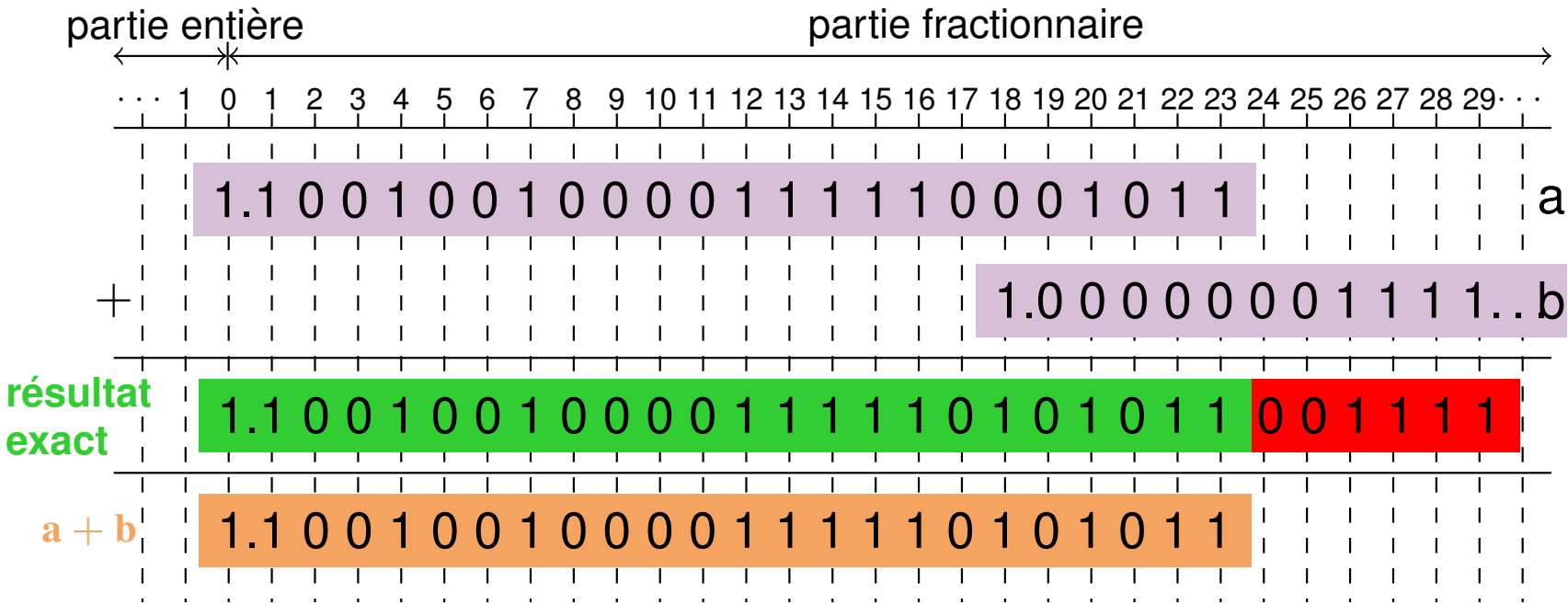
Lors de la soustraction de deux nombres très proches.



Si les opérandes sont eux-mêmes des résultats de calculs avec des erreurs d'arrondi, les 0 ajoutés à droite (partie rouge) sont faux. La cancellation est catastrophique quand il n'y a presque plus de chiffres significatifs.

Phénomène d'absorption

Lors de l'addition de deux nombres ayant des ordres de grandeur très différents où l'on peut « perdre » toute l'information du plus petit des deux nombres.



Combiner avec la *cancellation* ce phénomène peut tourner à la catastrophe. Par exemple en simple précision, si $a = 1$ et $b = 2^{-30}$ alors : $a \oplus b = 1$ et donc $(a \oplus b) \ominus a = 0$.

Exemple du phénomène d'absorption

On calcule numériquement, pour de grandes valeurs de N, la somme :

$$\sum_{i=1}^{N} \frac{1}{i}$$

Résultats d'un programme C (flottant SP) sur un processeur Pentium4 :

ordre	N			
	10^5	10^6	10^7	10^8
exacte	1.209015e+01	1.439273e+01	1.669531e+01	1.899790e+01
$1 \rightarrow N$	1.209085e+01	1.435736e+01	1.540368e+01	1.540368e+01
$N \rightarrow 1$	1.209015e+01	1.439265e+01	1.668603e+01	1.880792e+01

Quelques erreurs spectaculaires

- En 1982 à la bourse de Vancouver, les calculs sur un indice nouveau, de valeur initiale 1000.0, sont tronqués (plutôt qu'arrondis). Après 22 mois la valeur de l'indice calculé est de 520 au lieu de 1098.892 . . .
- Le 25 février 1991, à Dharan en Arabie Saoudite, un missile anti-missile Patriot américain rate l'interception d'un missile Scud iraquien. Conséquences : 28 morts, plus d'une centaine de blessés.
- Perte d'une plate-forme pétrolière en mer du Nord, au large de la Norvège, le 23 août 1991 (problème matériel + imprécision éléments finis) : coût estimé à 700 M$.
- Lors des élections dans le Schleswig-Holstein le 5 avril 1992, l'affichage du pourcentage de voix des Verts sans décimales puis avec deux décimales change le résultat et donne la majorité au SPD au parlement allemand.
- Le 4 juin 1996, lors de son premier vol, la fusée européenne Ariane 5 explose 30 secondes après son décollage causant la perte de la fusée et de son chargement estimé à 500 M$ (et 7 milliards de dollars d'investissement).
- Liens web :
  `http://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html`
  `http://www5.in.tum.de/~huckle/bugse.html`

L'explosion du vol Ariane 501

Le 4 juin 1996, lors de son premier vol, la fusée européenne Ariane 5 explose 30 secondes après son décollage causant la perte de la fusée et de son chargement estimé à 500 M$. Le système de guidage de la fusée s'est arrêté à la suite de l'arrêt des deux unités flottantes qui contrôlaient son programme.

Après deux semaines d'enquête, un problème est trouvé dans le système de référence inertiel : la vitesse horizontale de la fusée par rapport au sol était calculée sur des flottants 64 bits et dans le programme du calculateur de bord, il y avait une conversion de cette valeur flottante 64 bits vers un entier signé 16 bits.

Rien n'était fait pour tester que cette conversion était bien possible mathématiquement (sans dépassement de capacité).

Les tests ont été effectués pour Ariane 4 qui, étant moins puissante qu'Ariane 5, avait une vitesse horizontale suffisamment faible pour tenir sur un entier 16 bits ce qui n'était pas le cas d'Ariane 5.

Scud ! je l'ai raté !

L'horloge interne du missile Patriot mesure le temps en $1/10$s. Pour obtenir le temps en secondes le système multiplie ce nombre par 10 en utilisant un registre de 24 bits en virgule fixe.

$1/10$ n'est pas finiment représentable en base 2 et donc a été arrondi. L'expansion binaire de $1/10$ est $0.0001\underline{1100}$. Le registre de 24 bits contient $0.000110011001100110011001$ et introduit une erreur binaire de $0.000000000000000000000\underline{1100}$, soit 0.000000095 environ en décimal.

En multipliant ce chiffre par le nombre de $1/10$s en 100h (le temps écoulé entre la mise en marche du sustème et le lancement du missile Patriot), le temps écoulé est sous-estimé de $0.000000095 \times 100 \times 60 \times 60 \times 10 = 0.34$s.

Un Scud parcourt envion 1.676m/s, donc en 0.34s parcourt plus de 500m ce qui le fait largement sortir de la zone d'acquisition de sa cible par le missile d'interception Patriot.

Propriétés des flottants IEEE-754

- L'égalité n'est pas réflexive (NaN $\neq$ NaN).
- L'addition flottante et la multiplication flottante sont commutatives mais pas associatives (dépassements de capacité par exemple).
- La multiplication flottante n'est pas distributive par rapport à l'addition.
- Si aucun dépassement de capacité vers l'infini ou vers zéro ne se produit pendant les calculs, les propriétés suivantes sont vérifiées avec des nombres flottants et des opérations flottantes IEEE-754 :

$$x \oplus 0 = x \ominus 0 = x$$

$$x \otimes 1 = x \otimes 1 = x \qquad x \otimes -1 = x \oslash -1 = -x$$

$$2 \otimes x = x \oplus x = 2x \qquad 0.5 \otimes x = x \oslash 2 = x/2$$

$$\circ(\sqrt{x}) \geq 0 \text{ si } x \geq 0 \qquad \circ(\sqrt{-0}) = -0$$

Propriétés des flottants IEEE-754

Théorème de Sterbenz et résultats de Goldberg

Théorème de Sterbenz (1974)

Si x et y sont deux nombres flottants tels que $y/2 \leq x \leq 2y$ alors $\circ(x-y) = x-y$.

Goldberg, 1991

Sans dépassement de capacité (vers $+\infty$ ou vers 0) et de division par 0, pour deux flottants x et y on a :

$$-1 \leq \sqrt{\frac{x}{x^2+y^2}} \leq 1$$

en dépit des 5 opérations à effectuer, génératrices d'erreur

Références : What every computer scientist should know about floating-point arithmetic. David Goldberg. ACM Surveys. 1991

Raisonnement semi-automatisé

Directement dans un système d'aide à la preuve : Coq (Coupet-Grimal, Boldo, Zimmermann, Théry, Rideau, Filliâtre, etc.), PVS (Miner/Carreño (1996), Muñoz, NASA, Boldo), HOL light (Harrison, INTEL)

Avec une surcouche dédiée :

- Caduceus (Boldo, Filliâtre, Orsay : Coq[Why] / PVS)
- Zénon dans l'Atelier FOCAL (Doligez : Foc, Coq, Elan)
- Gappa (Melquiond, 2005)

Gappa : un outil prometteur

Il est capable de calculer des bornes sur des expressions arithmétiques en général et sur les erreurs d'arrondi d'un code numérique en particulier. Il génère aussi une preuve formelle de ces bornes.

- Analyse d'intervalles (Boost intervalles à bornes double, rationels GMP. dyadic MPFR) pour propager les retenues
- Base de théorèmes sur les bornes sur les valeurs arrondies et les erreurs d'arrondi
- Base de règles de réécriture pour réduire les intervalles calculés (réduction de décorrélation, etc.)
- Écrit C++ avec une syntaxe proche de celle du code C à valider et ne nécessite pas la connaissance d'un système de preuves généraliste
- Gappa génère un terme de preuve (Coq, HOL light et PVS ?) vérifiable indépendamment.

Outils d'analyse

Outils d'analyse de la précision des calculs :

- s'appuient tous sur l'analyse d'intervalles
  $(\nabla(x) \leq x \leq \Delta(x)$,
  $\nabla(x) + \nabla(y) \leq \nabla(x+y) \leq y \leq \Delta(x+y) \leq \Delta(x) + \Delta(y))$
- analyse dynamique : CADNA
  Control of Accuracy and Debugging for Numerical Applications, méthode CESTAC et arithmétique stochastique, Vignes et Chesneaux, LIP6, pour C, C++, Fortran et Ada
- analyse statique : Fluctuat
  CEA (Éric Goubault, Martel, Putot) interprètation abstraite, analyse d'intervalles, etc. pour C

Améliorations de l'arithmétique flottante

- norme étendue aux opérations élémentaires avec la règle de l'arrondi correct, CRlibm (de Dinéchin, Lauter, LIP Lyon, certifié correct par Gappa), Post Ultimate libm (Ziv, IBM Tel Aviv)
- double double et compensée (Veltkamp, Dekker (1971), Goldberg (1991), Rump et al.(2005))
- arithmétique adaptative de Ziv
- arithmétique d'intervalles (implémentations dans divers langages 3 pages écrans ; MPFI (Revol/Rouillier 2002), Boost : 2 parmi les 11 packages pour C++ ; effort d'intégration à la norme C++ à l'horizon 2010)
- arithmétique multi-précision en C : MPFR (LORIA, projet SPACES, arithmétique adaptative), scslib (opérations de la norme IEEE, LIP Lyon) ; d'intervalles multi-précision en C++ (MPFI, Revol & Rouillier) ; IRRAM en C++ (Norbert Müller 1997, arithmétique adaptative)

Le cadre théorique : les nombres réels constructifs ou calculables

- $\mathcal{R}$ est un corps commutatif archimédien, constructivement complet, clos pour les fonctions élémentaires.
- $\mathcal{R}$ est un sous-ensemble dénombrable de $\mathbb{R}$.
- Théorèmes d'indécidabilité Il n'existe pas d'algorithme général pour
  - comparer deux réels calculables (en particulier savoir si un réel calculable est nul ou non),
  - calculer la partie entière d'un réel calculable,
  - déterminer si un réel calculable est rationnel,

  Mais
- Il existe un algorithme de comparaison de deux nombres réels qui terminent si les deux nombres sont distincts.
- On peut faire des comparaisons à ε près
  $x =_\varepsilon y \iff |x - y| < \varepsilon$

Les différentes versions

- les fractions continues (Gosper 1972, Vuillemin 1987, Ménissier 1989, Lester 2001), les LFT (à la Potts, revu par Heckmann)
- les expansions infinies (Boehm 1987, Ménissier 1989)
- les suites de Cauchy récursives (Boehm 1987, Ménissier 1994) première description complète et prouvée, implémentation CREAL en Ocaml (Filliâtre), XR en python et C++, xrc en C (Briggs), une version « stochastique » MPFS qui s'appuie aussi sur MPFR (Briggs)
- les raffinements successifs flottants iRRAM (Norbert Müller 1997) (+analyse)