Tests exhaustifs des fonctions élémentaires

(recherche de pires cas pour l'arrondi exact)

Vincent LEFÈVRE

Journées au vert

15–16 mai 2003

Systèmes à virgule flottante

Caractéristiques : base b, taille de mantisse n, plage des exposants $E_{\min} \ldots E_{\max}$.

En général, $b = 2$ (mais 16 et 10 utilisés par certaines machines).

Nombre machine x : *signe* $s_x = \pm 1$, *mantisse* $m_x = x_0.x_1x_2\ldots x_{n-1}$, *exposant* $E_x \in [\![E_{\min}, E_{\max}]\!]$.

$$x = s_x \times m_x \times b^{E_x}$$

Par exemple :

- Simple précision : $b = 2$, $n = 24$, $E \in [\![-126, +127]\!]$.
- Double précision : $b = 2$, $n = 53$, $E \in [\![-1022, +1023]\!]$.

Modes d'arrondi

En général, la somme, le produit ou le quotient de 2 nombres machine *n'est pas* un nombre machine : il doit être *arrondi*.

Modes d'arrondi :

- vers $-\infty$: $\nabla(x)$ est le plus grand nombre machine $\leqslant x$;
- vers $+\infty$: $\triangle(x)$ est le plus petit nombre machine $\geqslant x$;
- vers 0 : $\mathcal{Z}(x)$ est égal à $\nabla(x)$ si $x \geqslant 0$, et à $\triangle(x)$ si $x < 0$;
- au plus près : $\mathcal{N}(x)$ est le nombre machine le plus proche de x.

Arrondi exact

- $+, -, \times, /, \sqrt{}$: norme IEEE 754-1985 : arrondi du résultat exact (appelé *arrondi exact*).
- Fonctions élémentaires ($\exp$, $\log$, $\sin$, $\cos$, etc.) : aucune exigence. Implémentation parfois peu précise.

Avantages de l'arrondi exact :

- implémentations plus précises,
- portabilité (réduction des coûts des logiciels),
- construction et preuve d'algorithmes,
- arithmétique d'intervalles.

Le dilemme du fabricant de tables

Problème : garantir l'arrondi exact des fonctions élémentaires *à coût raisonnable*.

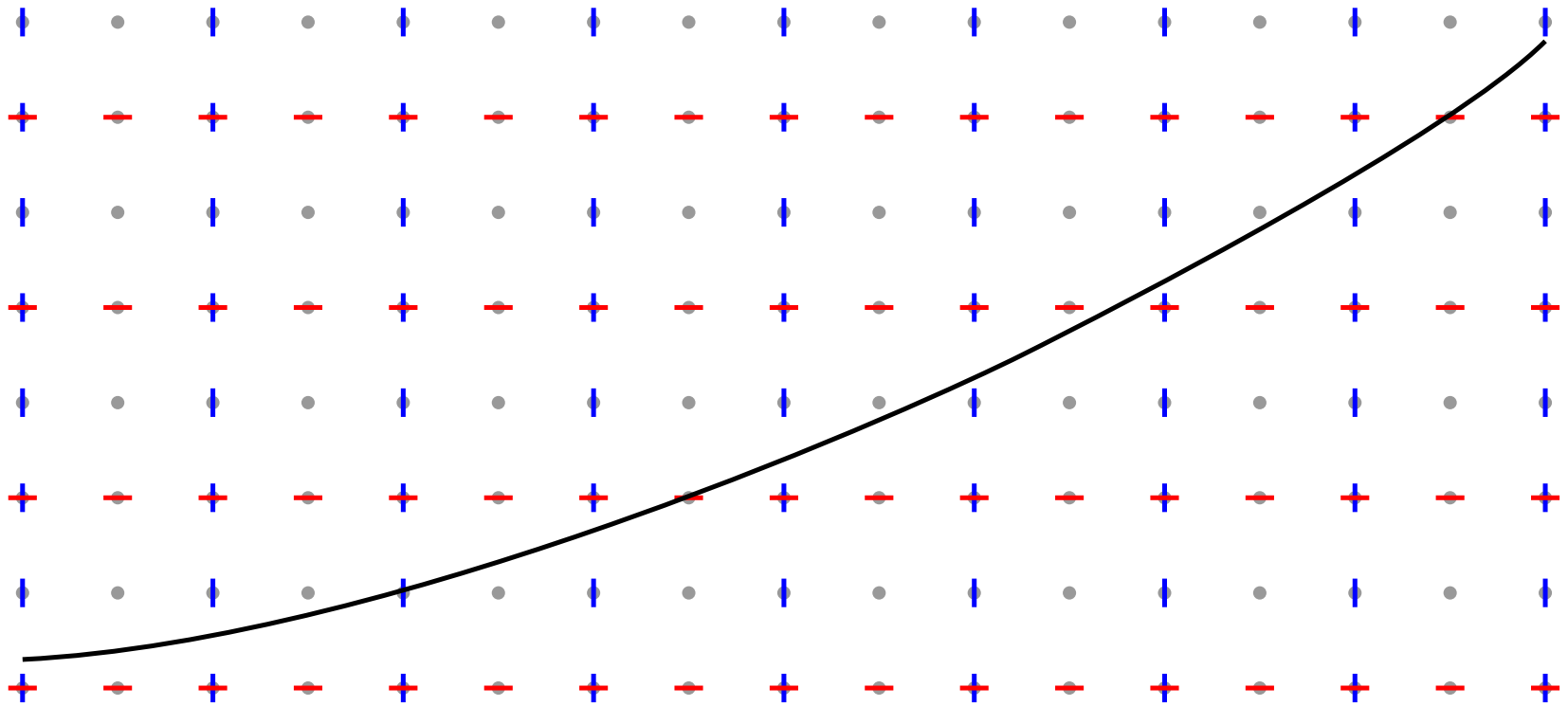
- Calcul d'une valeur approchée de $f(x)$ à ε près.
- Ce résultat peut être *a priori* très proche d'une frontière entre deux arrondis possibles : distance ε'.
- Si $\varepsilon' < \varepsilon$, impossible de garantir l'arrondi exact.

$\rightarrow$ À quelle précision m faut-il faire les calculs intermédiaires ?

Application d'un théorème de Nesterenko et Waldschmidt (1995)
$\rightarrow$ bornes de l'ordre de plusieurs millions ou milliards de bits.

Seule solution acceptable : tests exhaustifs.

Rappel du problème de recherche de pires cas



Nouveaux tests

But : pouvoir rapidement refaire les tests effectués il y a quelques années

- avec correction des bugs qui ont été découverts,
- en tenant compte de l'évolution des machines et des réseaux,
- avec des mesures des temps de calculs,

en attendant une réécriture complète.

Modifications par rapport aux anciens tests

Réécriture en C ISO des routines qui étaient écrites en assembleur Sparc, avec utilisation de GMP pour les calculs en multiprécision, sans changer la taille des variables (pour être certain de ne pas introduire de bug à ce niveau).

Algorithme de minoration de la distance d'un segment de droite à $\mathbb{Z}^2$ ($\sim$ algo d'Euclide) : version entièrement soustractive $\rightarrow$ divisions.

- Division (opération très lente) uniquement quand le quotient est suffisamment grand (test à l'aide d'un décalage, donc rapide), en tenant compte du fait qu'en général, le quotient est très petit.
- Les dépassements de capacité sur les entiers sont évités.

```
if (LOGMS && (y >> LOGMS) > x)
  {
    uint64_t q = y / x;
    TESTEND(q >= N);  /* avoid overflow below */
    y -= (unsigned int) q * x;
    u += (unsigned int) q * v;
  }
else
  while (x < y)
    {
      TESTEND(u + v >= N);
      y -= x;
      u += v;
    }
```

Nouveaux scripts permettant de générer les batches, les lancer, et récupérer les résultats.

- Écriture en sh, mais trop de problèmes (incompatibilités, bugs).
- Finalement, Perl + un peu de zsh.

Maple toujours utilisé... mais Maple 5 ne tourne plus sous les nouvelles versions de Linux, et Maple 6 et 7 ont de gros bugs.
→ Maple 8.
Arithmétique d'intervalles : intpakX au lieu de intpak (buggé).

→ Génération des sources C (première partie de la première étape) sur une seule machine et non plus en parallèle avec le reste de la première étape.

Machines utilisées

- Cluster de calcul du CCH (8 machines biprocesseurs) :
  14 976 heures.
- Cluster du LIP / ENS-Lyon (3 machines quadriprocesseurs) :
  12 016 heures.
- Diverses machines de MEDICIS :
  8 678 heures.
- Machines SPACES au Loria (+ ay) :
  2 049 heures.

(Temps de calcul pour l'étape 1.b uniquement.)

Fonctions actuellement testées

- 2^x, exposants -1 à 9 (en fait, 0, 1, et 5 entre 32 et 33) ;
- $\log_2(x)$, exposants -1 et 0 ;
- $\exp(x)$ et $\exp(-x)$, exposants -1 à 9, valeurs restreintes à la double précision IEEE 754 ;
- $\log(x)$, exposants -1 et 0 ;
- $1/x^2$, exposant -1 ;
- $\sin(x)$, exposants -4 à 0.

Pire cas

Toujours le même... Pour

$$x = 0.01111111100111011001110111001110011101000011110110 1101$$
$$= \frac{8980155785351021}{18014398509481984},$$

$\sin x$ est égal à :

$$0.01111010011001010100000111001100001100010001101001010 1\,1^{65}\,0000\ldots$$

En considérant la réciproque, on a : pour

$$x = 0.01111010011001010100000111001100001100010001101001011 0$$
$$= \frac{4306410053968715}{9007199254740992},$$

$\arcsin x$ est égal à :

$$0.01111111100111011001110111001110011101000011110110 1101\,0\,0^{64}\,1000\ldots$$

Comparaison des temps de calcul par intervalle

Fonction $\exp(x)$, exposant 0, découpage en $2^{13} = 8192$ sous-intervalles, cluster du LIP :

intervalle	temps
2942	66.94
2943	52.94
2944	73.19
2957	72.65
2958	85.57
2959	73.73

$\rightarrow$ irrégularités. Cause exacte ?

Comparaison des temps entre les étapes 1.a, 1.b et 2

Avec des découpages en $2^{13} = 8192$ sous-intervalles, et plus si la fonction s'approche mal par un polynôme de degré 2 :

- Étape 1.a : 2 à 3 secondes par intervalle.
- Étape 1.b : 80 à 200 heures au total si la fonction s'approche assez bien ; plusieurs milliers d'heures pour $\exp(x)$, exposant 8, par exemple. Cette étape est parallélisée.
- Étape 2 : quelques secondes par intervalle.

Si la fonction s'approche bien, ce sont les étapes 1.a et 2 (actuellement non parallélisées) qui sont limitantes.

Si la fonction s'approche mal : tester SLZ...

Pour les futurs tests...

Nouveaux langage et outils :

- Preuves (majorations d'erreur) intégrées au langage et *vérification automatique*.
- Uniformisation simple précision / longlong.h / multiprécision (et entiers / virgule fixe / virgule flottante), sur les opérateurs *et* les données.
- Génération de code C (voire assembleur) ou autre.
- Possibilité de mesure (nombre d'opérations, mémoire, etc.).
- Possibilité d'imbriquer calculs et génération de code (cf première étape).

Implémentation en XML (en utilisant MathML) / XSLT.

Avantages :

- Plus besoin d'utiliser Maple.
- Toutes les étapes en parallèle.
- Certaines limitations (e.g. entiers de base du langage C) ne sont plus naturelles et sont ainsi supprimées.
- Intégration plus facile de nouveaux algorithmes.