Tests exhaustifs des fonctions élémentaires

(recherche de pires cas pour l'arrondi exact)

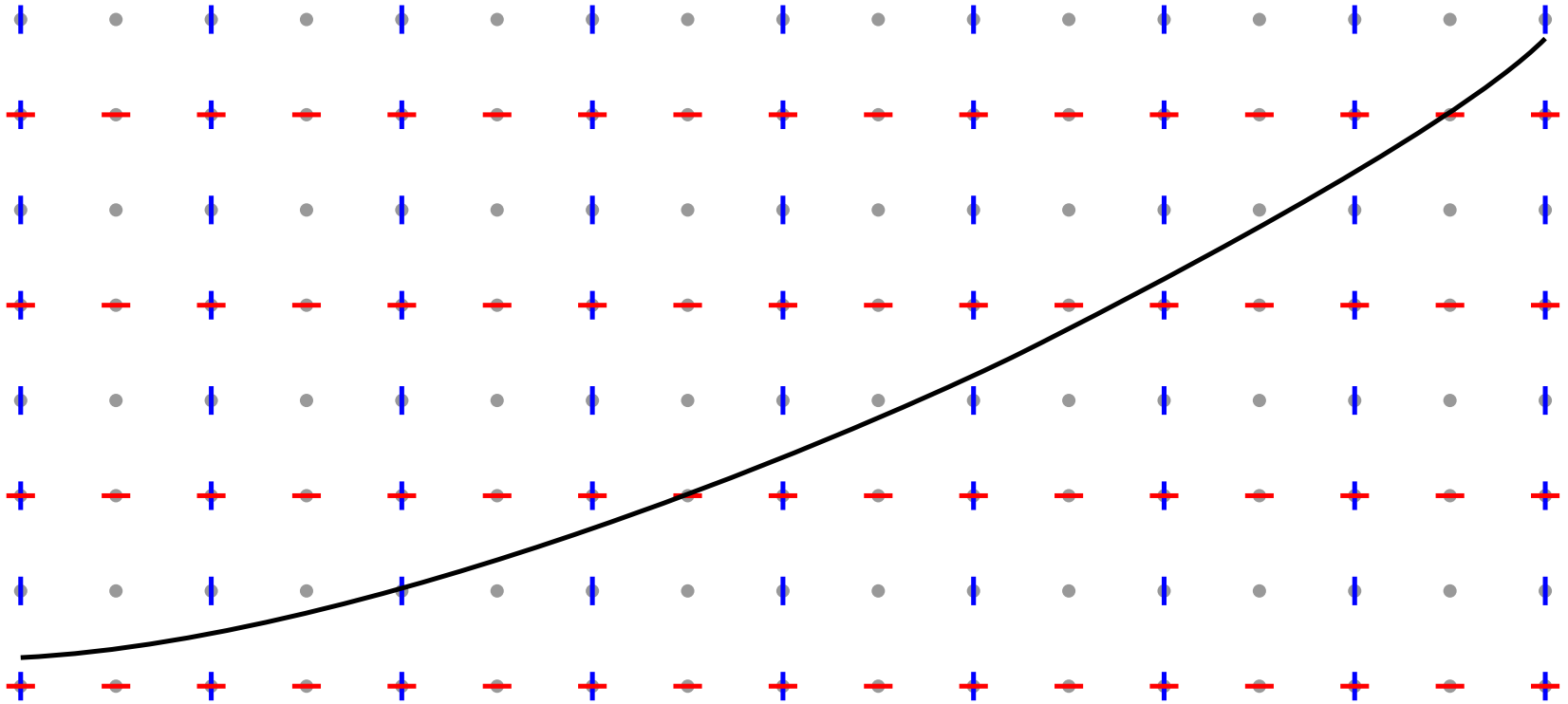
Vincent LEFÈVRE

Loria / INRIA Lorraine

Journées Arinews / ASAO

17–18 novembre 2003

Problème de recherche de pires cas



Historique

1996–2001 : tests en 2 étapes (filtre + test des valeurs non filtrées) utilisant

- Perl pour le contrôle (serveur + clients) et la génération de code ;
- Maple + intpak pour l'approximation de la fonction testée par un polynôme (de grand degré) et pour la deuxième étape ;
- assembleur Sparc pour le filtre (dans la première étape).

Problèmes avec les anciens programmes de test

– Il n'y a plus beaucoup de machines Sparc et ce ne sont plus les plus rapides ;
– très lents si la dérivée de la fonction s'approche très bien d'un rationnel simple (e.g. nombreuses fonctions dans les intervalles où l'exposant est très petit) ;
– contrôle de l'optimisation difficile ;
– parallélisation (serveur + clients) limitée à un seul réseau ;
– fiabilité ?

→ Modifications nécessaires à court terme, réécriture à long terme.

Nouveaux tests

Janvier 2003 – ? ?

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).
- Les dépassements de capacité sur les entiers sont évités.

Mon ancien algorithme

Initialisation : $x = \{a\}\ ;\ y = 1 - \{a\}\ ;\ d = \{b\}\ ;\ u = v = 1;$

Boucle infinie :

- **si** $(d < x)$
  - **tant que** $(x < y)$
    - **si** $(u + v \geqslant N)$ **fin**
    - $y = y - x;\ u = u + v;$
  - **si** $(u + v \geqslant N)$ **fin**
  - $x = x - y;\ v = v + u;$
- **sinon**
  - $d = d - x;$
  - **tant que** $(y < x)$
    - **si** $(u + v \geqslant N)$ **fin**
    - $x = x - y;\ v = v + u;$
  - **si** $(u + v \geqslant N)$ **fin**
  - $y = y - x;\ u = u + v;$

Valeur de retour : d

Une partie de la réécriture...

```
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 tests (suite)

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.

Fonctions actuellement testées (double précision)

- $\exp(x)$ et $\exp(-x)$, exposants de -1 à 9 (jusqu'à `DBL_MAX`) ;
- $\log(x)$, exposants -1 et 0 ;
- 2^x, exposants de -1 à 9 (en fait, 0, 1, et 5 entre 32 et 33) ;
- $\log_2(x)$, exposants -1 et 0 ;
- 10^x, exposants de -2 à 8 (jusqu'à `DBL_MAX`) ;
- 10^{-x}, exposants de -2 à 0 (à terminer...) ;
- $\log_{10}(x)$, exposants -1 et 0 ;

- $\sin(x)$, exposants de -25 à 0 + une partie de 1 ;
- $\cos(x)$, exposants de -25 à -1 + une partie de 0 ;
- $\tan(x)$, exposants de -25 à -4 ;
- $\arctan(x)$, exposants de -4 à 0 ;
- $\sinh(x)$, exposants de -25 à 9 ;
- $\cosh(x)$, exposants de -25 à 5 (inutile d'aller plus loin) ;
- $1/x^2$, exposant -1 (précision de 19 bits à 58 bits).

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 ($f \rightarrow$ polynôme) : 2 à 3 secondes par intervalle.
- Étape 1.b (filtre) : 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 (tests avec Maple) : 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...

Machines utilisées et temps de calcul

- Cluster de calcul du CCH (8 machines biprocesseurs) :
  49 120 heures.
- Cluster du LIP / ENS-Lyon (3 machines quadriprocesseurs) :
  22 167 heures.
- Diverses machines de MEDICIS :
  8 885 heures.
- Machines SPACES au Loria (+ ay) :
  2 870 heures.

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

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 ?

```
  1.4  isq -1 44 32
  2.2  isq -1 45 33
  3.6  isq -1 46 34
  5.9  isq -1 47 35
  8.9  isq -1 48 36
 13.1  isq -1 49 37
 20.2  isq -1 50 38
 33.8  isq -1 51 39
 63.8  isq -1 52 40
 58.3  isq -1 53 40
 52.2  isq -1 54 40
 49.7  isq -1 55 40
 48.2  isq -1 56 40
 92.8  isq -1 57 41
185.3  isq -1 58 42
369.3  isq -1 59 43
```

Quelques anciens pires cas (en double précision)

Pour $x = 0.01111111100111011001110111001110011101000011101101101$:

$$\sin x = 0.01111010011001010100001110011$$
$$000011000100011010010101\ 1\ \underbrace{1111...1111}_{65\text{ bits}}\ 00...$$

($m = 53 + 1 + 65 = 119$, arrondi dirigé)

Pour $x = 0.01111010011001010100001110011000011000100011010010110$:

$$\arcsin x = 0.01111111100111011001110111001 1$$
$$= 10011101000011110110110 1\ \underbrace{0000...0000}_{64\text{ bits}}\ 10...$$

($m = 53 + 1 + 64 = 118$, arrondi dirigé)

Quelques nouveaux pires cas (en double précision)

$x = 1.11100001001011010110011001110100010011111111110000001 \times 2^{429}$:

$$\log_{10} x = 10000001.01101010011111010100110$$
$$1110010100111110010001001\underbrace{0000\ldots0000}_{68\text{ bits}}10\ldots$$

($m = 53 + 1 + 68 = 122$, arrondi au plus près)

$x = 1.11001110010000011101100011111010011001100101111111010 \times 2^{4}$:

$$\log_{10} x = 1.01110101111101001001110001101$$
$$0101101001110111010110\underbrace{0000\ldots0000}_{66\text{ bits}}10\ldots$$

($m = 53 + 1 + 66 = 120$, arrondi dirigé)

Tests de la libm (% de cas mal arrondis)

Proc OS	P Li	In F	In O	At Li	VE Li	A6 Li	A6 F	A5 Os	A6 Os	I1 Li	I2 Li	Sp So
exp	49	49	49	49	49	49	50	51	51	49	43	50
log	43	43	43	43	42	30	30	31	31	30	52	30
exp2	48			48	48	51				52	52	47
log2	45			44	44	50				50	50	49
exp10	52			52	52						66	51
log10	45	45	45	46	47		48	42	42		46	48
sinh	49	51	51	49	49	51	51	47	47	51	49	50
asinh	52	49	82	51	50	50	50	47	46	50	50	50
cosh	50	46	46	50	50	46	47	45	45	46	40	46
acosh	41	49	86	41	41	48	48	51	50	48	49	49
sin	51	51	51	51	51	49	49	51	51	49	51	49
asin	49	50	51	49	49	48	47	47	47	47	50	47
cos	49	49	49	49	49	45	45	50	50	45	50	50
acos	83	50	95	83	83	30	29	41	41	28	41	29
tan	52	52	52	52	51	51	51	49	49	50	46	51
atan	50	50	50	50	50	51	51	46	46	51	46	50
isq	55	53	53	55	55	53	53	48	48	53	55	57
isqrt	51	51	51	51	51	51	51	55	55	51	54	53

Proc OS libm	PA-RISC Linux 2.2.5	Alpha EV6 Linux 2.3.1	PPC G4 Linux 2.3.2
exp	0	0	0
log	0	0	5 (régr.)
exp2	51	51	52
log2	50	50	50
exp10	66	66	66
log10	47	47	48
sinh	45	45	45
asinh	50	50	50
cosh	45	45	45
acosh	49	49	49
sin	0	0	0
asin	0	0	0
cos	5 cas faux	5 cas faux	0
acos	17	17	17
tan	0	0	0
atan	0	28% faux	0
isq	0	0	0
isqrt	0	0	0