

配列整数演算の精度、性能評価

濱口信行

nobuyuki.hamaguchi.sa@hitachi.com

(株)日立製作所 ソフトウェア事業部

概要

数値計算で使用される浮動小数点演算では、演算誤差を伴う為に、低い精度の計算結果をもたらす事がある。この問題に対し、4倍精度以上の演算を行う多倍長ライブラリを試作し、精度、性能評価を行った結果、浮動小数点加減算を整数演算に置き換えた配列整数演算は非常に有効で、応用範囲が広い事が、判明した。

Evaluation of accuracy and performance of Array-integer operation

Nobuyuki Hamaguchi

Software Division, Hitachi, Ltd.

Abstract

Floating point operations used in numeric calculation may result in lower precision because of errors in operations.

We have developed the Multi-precision library that calculates floating point data of quadruple or higher precision using the Array-integer operation, which replaces addition and subtraction of floating point data to arithmetic operation of integer data.

We have evaluated the accuracy of computed values and the performance, and come to the conclusion that Array-integer operation is very effective and is applicable for a wide area.

1. はじめに

数値計算で使用される演算は大別すると、浮動小数点演算と整数演算があるが、除算を除けば、誤差を発生させるのは前者である。また精度を向上させれば一般的に性能は低下するので性能評価も必要となる。本論文では、浮動小数点演算に整数演算を使用し、本課題に適用した結果を述べる。論文では、浮動小数点演算に整数演算を使用して行う事を配列整数演算としている。

なお、精度、性能評価のため浮動小数点数の表現形式はIEEE754形式をとり、 $P \geq 3$ のP倍精度として符号

部 1ビット、指数部 15ビット、仮数部 32* p -16 ビットを仮定している。

また、性能評価に使用したCPU、OS、カーネル、コンパイラの環境は以下で、1CPUで実行している。

Intel® Xeon 3.06GHz (2cpu L2キャッシュ 512KB/cpu),
Os: Red Hat Enterprise Linux ES, release 3
(taroon)

Kernel: 2.4.21-4 Elsmp on an i686

Intel Fortran V8.0 -02 (注-1)

(注-1)

IntelおよびIntel Xeonは、アメリカ合衆国およびその他

の国におけるインテルコーポレーションまたはその子会社の商標または登録商標です。

Red Hatは、米国およびその他の国でRed Hat, Inc. の登録商標若しくは商標です。

Linuxは、Linus Torvaldsの米国およびその他の国における登録商標あるいは商標です。

2. 誤差要因

数値計算で使用される浮動小数点演算の誤差は、

(1) 加減算での桁落ち、情報落ち

(2) 乗算、除算の丸め誤差

があるが、加減算、乗算での誤差は配列整数演算で防ぐ事ができ、除算はより高精度の逆数演算で精度の向上が図れる。

3. 浮動小数点数の表現

0, および特異値($-\infty$, ∞ , NaN等) は、演算の必要がない事や、演算前の段階でわかるため、演算時は、特異値はないとし、0は演算時に判定すれば、通常演算処理に寄与しないので、配列整数テーブルで表現するのは、0以外の通常の浮動小数点数Aとする。この場合Aは、 $A = \pm P * 2^{**}(-36000)$ と表す事ができる。(P:は正整数)。符号値は別にもち、正整数を整数配列で表わす様にする。

今、整数型4バイト整数配列 IA(2400) をとり、各配列要素32ビットの内30ビットを使用するものとすれば、正整数Pは $P = IA(1) * 2^{**}(30 * (2400 - 1)) + IA(2)$

$$* 2^{**}(30 * (2400 - 2)) + \dots + IA(J) * 2^{**}(30 * (2400 - J)) + \dots + IA(2400)$$

と一意に表現できる。サイズ2400は以下の様な演算途中のオーバーフロー、アンダーフローの影響を防ぐ様に定めている。

演算に使用される変数の値と結果は、浮動小数点数が表現される値で途中の演算でオーバーフローする例を図1に示した。

4. 総和演算

総和計算

T=0

DO I=1, N

T=T+A(I)

END DO

では、A(I)に0があった場合は、加算操作を行わない。0以外の浮動小数点数を整数配列IA(1200)で表現し、演算も整数演算を使用する様にすれば、演算中にオーバーフロー、アンダーフローは発生せず、誤差なく結果を得る

事が出来る。

$$Pi = Ai * 2^{**}(n) \quad Pi: \text{整数} \quad T = \sum Ai = (\sum Pi) / 2^{**}(n) \quad (n=18000)$$

たとえば、4倍精度で以下の計算をする場合を考える。

$$a=77617.0 \quad b=33096.0$$

$$f = 333.75 * (b^{**}6) + (a^{**}2) * [11 * (a^{**}2) * (b^{**}2) - (b^{**}6) - 121 * (b^{**}4) - 2] + 5.5 * (b^{**}8) + a / (2 * b) \quad [1]$$

通常の計算では

1. 17260394005317863185883490452018 の結果が得られる。

ところが、この計算の理論値は、 $-54767/66192$

$= -0.827396059946821368141165095497981370$ である。そこで、

$$A(1) = 333.75 * (b^{**}6)$$

$$A(2) = (a^{**}2) * 11 * (a^{**}2) * (b^{**}2)$$

$$A(3) = -(b^{**}6) - 121 * (b^{**}4) - 2$$

$$A(4) = 5.5 * (b^{**}8) + a / (2 * b)$$

$$A(5) = -54767 / 66192$$

$$A(6) = 33096.0$$

$$A(7) = a / (2 * b)$$

とし、総和を取ると、

$$\text{answer} = -0.8273960599468213681411650954979816$$

となり、10進31桁結果が一致する。

また、総和演算での、整数テーブルを持ち、整数演算を行う方法は、精度のみならず、多くの要素の和を求める場合には、高性能となる(図2)ため、演算時間の増大という問題も解消される。

5. 内積演算

内積計算

T=0

DO I=1, N

T=T+A(I)*B(I)

END DO

において、Tの精度はA(I)*B(I)の持つ精度で定まるので、丸め処理を行わない乗算にすれば精度の向上が図れる。乗算A*Bにおいては、以下の方法で誤差なく計算できる。

$$A = (-1) ** S1 * 2^{**} T1 * (1. F)$$

$$B = (-1) ** S2 * 2^{**} T2 * (1. G) \quad \text{とすると、}$$

$$C = A * B = (-1) ** S * 2^{**} T * (1. H) \quad \text{では、}$$

$$S = \text{MOD}(S1 + S2, 2)$$

$$T = T1 + T2 \quad 1 \leq (1. F) * (1. G) < 2$$

$$T = T1 + T2 + 1 \quad 2 \leq (1. F) * (1. G) < 4$$

mを仮数部のビット数とすると、 $2^{**}m * (1. F) = 2^{**}m + 1$ ($0 \leq m \leq 2^{**}m - 1$)

$2^{**m}*(1.G)=2^{**m+n}$ ($0 \leq n \leq 2^{**m}-1$)
 1, nは正の整数で、浮動小数点データの仮数部分から得られる。

$2^{**2m}*(1.F)*(1.G)=(2^{**m+1})*(2^{**m+n})$ より、
 $1 \leq (1.F)*(1.G) < 2$ は $2^{**2m} \leq (2^{**m+1})*(2^{**m+n}) < 2^{**2m+1}$
 $2 \leq (1.F)*(1.G) < 4$ は $2^{**2m+1} \leq (2^{**m+1})*(2^{**m+n}) < 2^{**2m+2}$
 と同値関係にある。

符号部、指数部、整数乗算結果が2^m以内あるので、これにより誤差なく浮動小数点結果が得られる。

この事を内積演算精度検証プログラムを作成し、確認した。

プログラムの

```
DO I=1, N
  A(I)=A-B*I
  B(I)=(A-B*I)*(-1)**MOD(I+1, 2)
END DO
```

```
T=0
DO I=1, N
  T=T+A(I)*B(I)
END DO
```

結果の理論値(Nは2の倍数)

$M=N/2$
 $T=B*M*(2*A-B(1+2*M))$

Case1

A=2 B=2**(-61) N=10000000

理論値

0.867361737987463151631264862623795E-11

4倍精度

0.867361737987463152477842277159447E-11

4倍精度(本方式)

0.867361737987463151631264862623795E-11

Case2

A=2 B=2**(-71) N=10000000

理論値

0.847032947254299442237215649695000E-14

4倍精度

0.847032947254299445783495694991909E-14

4倍精度(本方式)

0.847032947254299442237215649695000E-14

Case3

A=2 B=2**(-91) N=10000000

理論値

0.807793566946316088740794387709507E-20

4倍精度

0.807793566946316088741610050849573E-20

4倍精度(本方式)

0.807793566946316088740794387709507E-20

また、内積演算は、多くの要素を扱う場合には、高性能(図3)の為、演算時間の増大という問題も解消される。

6. 除算方式

浮動小数点除算は、 $A/B=A*(1/B)$ とAとBの逆数の乗算で行われる。浮動小数点数の逆数を求める方法としては、テーブルや低精度な演算で求めた値を初期値C0とし、反復計算 $C_{n+1}=C_n \times (2-B \times C_n)$ により、1/Bを求める方法が一般的である。ただこの方式では、高精度になると、反復回数が多くなる事、反復計算時の加減算、乗算での丸め誤差、桁落ちに配慮する必要があり、演算速度も遅くなる。ここでは、除算 $A/B=A*(1/B)$ に対して直接計算で1/Bを求める方式を採用した。試作した多倍長ラブリでは、1/Bのみの値を計算するプログラムも多々ある為、逆数計算ルーチンを独立させ、B=0の場合の処理は、除算ルーチンの中で行う様にしている。

性能面では、loop length=1,000,000での性能測定結果を図4から図7に示した。

測定結果より、高精度になると、逆数計算の演算時間は乗算の演算時間と同程度となり、除算は乗算の2倍程度の実行時間となり、高速になっている。

高精度浮動小数点除算の適用例として3つの場合を示した。

(1) sin関数の計算

4倍精度でTaylor展開式より $\sin(x)$ を計算する事を考える。

$$\sin(x) = x - x^{**3}/3! + x^{**5}/5! - x^{**7}/7! + \dots$$

は $|x| < \infty$ で収束するが $|x|$ が大きいと収束が遅く精度も悪い。このため、この計算では、周期性を利用して、引数の範囲を $|x| < \pi/2$ となる様にして、 $t = [|x| / (\pi/2)]$ $t = |x| - 1 * \pi/2$ でtを求め、 $\sin(x)$ の値を計算する。[3]

ここで単純にこのまま計算すると、引数xは4倍精度分の精度を持つが、tは4倍精度分の精度はない。この為、 $|x|$ が大きいと、計算せず、メッセージを出す処

理系もある。この様な場合、
 $|x|/(\pi/2)$ 、 $|x|-1*\pi/2$ を8倍精度で計算する事により、 t が4倍精度分の精度が持てる
 範囲も大きくなり、通常の引数範囲でも、より良い精度の計算結果を得る事が出来る。

(2) 特殊関数の計算

科学技術計算では、spence 関数の様な特殊関数が使用される事が多々ある。[4]

$$Sp(z) = \int_0^z \frac{t}{(\exp(t)-1)} dt = \sum_{n=1}^{\infty} B_n (-\ln(1-z))^{n-1} / (n+1)!$$

[5] の右辺の有限値で計算するが、8倍精度の様な高精度の関数値を作成するためには、 n の大きなベルヌーイ数 B_n 、 $B_n(n+1)!$ の整数除算が必要となる。仮数部の大きい浮動小数点表現は大きな値の整数を表現出来る。(仮数部240ビットなら $2^{240} \approx 10^{72}$) ため、 B_n の

$$\text{循環式 [6]} \\ \binom{2n+1}{j} * B_1 \binom{2n+1}{j} + B_2 + \dots + (-1)^{n-1} \binom{2n+1}{2n} * B_{n-n-1/2}$$

で、高精度での spence 関数に用いる係数を算出する事が出来る

(3) 多重積分

2次元積分 $I = \int_0^1 \int_0^1 f(x,y) dy dx$ を計算する場合、
 優良格子点法により求める方法がある

$$I_n = 1/N \sum_{0 \leq k \leq N-1}$$

$$f(\{ g_1(N)/N * k \}, \{ g_2(N)/N * k \}) \quad [7]$$

ここで、 $F_1 = F_2 = 1$ $F_n + 2 = F_{n+1} + F_n$ ($n=1, 2, \dots$)

(Fibonacci 数列)

$N = F_n$ $g_1(N) = 1$ $g_2(N) = F_{n-1}$ で $\{x\}$ は x の小数部分を表している。精度よく計算するためには、 N を大きくとる事と $\{x\}$ を高精度に計算しなければならぬ。 $\{x\}$ を高精度に計算するには、 $g_2(N) * k / N$ で剰余を計算するが、大きな整数の除算になるため、多倍長浮動小数点除算を使用する。

7. 異なる形式の表現形式への適用

多倍長浮動小数点数の τ - t 表現には同じ精度の浮動小数点 τ - t 複数個で表す方法がある。[2]

例えば、8倍精度=4倍精度+4倍精度

$$= \text{倍精度} + \text{倍精度} + \text{倍精度} + \text{倍精度}$$

などである。この場合、総和、内積演算では、配列整数演算で、

$$T = \sum A_i \quad (i=1, n) = \sum_{(i=1, n)} \sum_{(j=1, 2)} A_{ij}$$

4倍精度+4倍精度

$$= \sum_{(i=1, n)} \sum_{(j=1, 4)} A_{ij}$$

倍精度+倍精度+倍精度+倍精度

$$T = \sum_{(i=1, n)} (A_i * B_i) = \sum_{(i=1, n)}$$

$$(\sum_{(j=1, 2)} \sum_{(k=1, 2)} (A_{ij} * B_{ik}))$$

4倍精度+4倍精度

$$= \sum_{(i=1, n)} (\sum_{(j=1, 4)}$$

$$\sum_{(k=1, 4)} (A_{ij} * B_{ik}))$$

倍精度+倍精度+倍精度+倍精度

除算、平方根でも丸め誤差のない乗算を使用する事により、

$A/B = T1 + T2$, $\text{SQRT}(A) = T1 + T2$ ($A, B, T1, T2$ は同じ精度) と表す事が出来る。

2つの計算例を以下に示した。

(1) 除算 : 4倍精度-8倍精度

$$1 + 1/2 + 1/3 + \dots + 1/n \quad (n=1000000)$$

4倍精度

$$0.1439272672286572363138112749318902E+02$$

4倍精度+4倍精度

$$0.1439272672286572363138112749318859E+02$$

8倍精度

$$0.1439272672286572363138112749318859E+02$$

8倍精度の誤差情報で4倍精度+4倍精度は8倍精度と仮数部210ビットまで等しくなっている。

(2) 平方根: 4倍精度-8倍精度

$$\sqrt{1 + \sqrt{2} + \sqrt{3} + \dots + \sqrt{n}} \quad n=1000000$$

4倍精度

$$0.6666671664588221083559787667952103E+09$$

4倍精度+4倍精度

$$0.6666671664588221083559787667951932E+09$$

8倍精度

$$0.6666671664588221083559787667951932E+09$$

8倍精度の誤差情報で4倍精度+4倍精度は8倍精度と仮数部212ビットまで等しくなっている。

8. まとめ

以上の検討により、配列整数演算は、演算時間の増大も押さえられ、誤差の解析、精度改善が行える事、配列整数 τ - t プールは、異なる表現形式を持つ処理系にも適用できその応用範囲の広い事がわかったが、今後は、更なる性能面での向上を検討して行く。

9. 参考文献

- [1] A review on Interval Computation -Software and Applications
Chenyi Hu, Shanying Xu, and Xiaoguang Yang
- [2] Octuple-precision floating point on Apple G4
R. Crandall, Advanced Computation group, Apple Computer And
J. Paradooulos, University of Maryland College Park
- [3] 山内二郎, 宇野利雄, 一松信 電子計算機のため
数値計算法 III 培風館 1981
- [4] Yoshimitsu Shimizu
Exact calculation of the heavy quark-pair
production in $O(\alpha_s)$ In two-photon process
Department of Physics, KEK, 1996
- [5] J. Fujimoto, M. Igarashi, N. Nakazawa, Y. Shimizu and K. Tobimatsu,
Progress of Theoretical Physics,
Supplement No. 100 (1990) 1-379
- [6] 高木貞治 解析概論 岩波書店 1969
- [7] 杉原正顯, 室田一雄 数値計算法の数理
岩波書店 2003

```
a1=(2.0-0.5**51)*2.0**515
a2=(2.0-0.5**50)*2.0**515
b1=(2.0-0.5**51)*2.0**515
b2=(2.0-0.5**50)*2.0**515
s=a1*b1-a2*b2
write(6,1) s
1 format(1h ,2x,'s=',c25.17)
stop
end
```

s=-Infinity (通常演算 倍精度)
s= 0.20437404769635524+296 (指数部拡張 4倍精度)

(図1) 引数、結果ともに表現可能な数値で、演算途中で
オーバーフローする例

10. 謝辞

精度検証にあたり御指導いただきました 高工機
ギ-加速器研究機構 藤本、石川、金子先生、及び総合
研究大学院大学の清水先生に感謝致します。

