社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－172 (23)
2007－HPC－109 (23)
2007／3／1

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 ｿﾌﾄｳｴｱ事業部

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ﾌﾞﾗﾘを試作し、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-inteter 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≧3 のP倍精度として符号部 1ﾋﾞｯﾄ、指数部 15ﾋﾞｯﾄ、仮数部 32*p-16 ﾋﾞ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、ｺﾝﾊﾟ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 Enterprose Linux ES release 3 (taroon)
Kernel: 2.4.21-4 Elsmp on an i686
Intel Fortran V8.0 -O2 (注-1)

(注-1)
IntelおよびIntel Xeonは，アメリカ合衆国およびその他

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(-∞, ∞,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、配列整数ﾃｰﾌﾞﾙで表現するのは、0以外の通常の浮動小数点数Aとする。この場合Aは、A=± 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ﾊﾞｲﾄ整数配列 IA(2400) をとり、各配列要素 32 ﾋﾞｯﾄの内 30 ﾋﾞ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))+.....+IA(J)*2**(30*(2400-J))
+.....+IA(2400)
と一意に表現できる。ｻｲｽﾞ 2400 は以下の様な演算途中のｵｰﾊﾞｰﾌﾛｰ、ｱﾝﾀﾞ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ﾊﾞ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、演算中にｵｰﾊﾞｰﾌﾛｰ、ｱﾝﾀﾞ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) Pi:整数 T=ΣAi= (Σ Pi)/2**(n) (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 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) [1]
```

通常の計算では
1.172603940053178631858834904520 18 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)=-(a**2)*(b**6)
A(4)=-(a**2)*(121*(b**4))
A(5)=-(a**2)*2
A(6)=5.5*(b**8)
A(7)=a/(2*b)
```

とし、総和を取ると、

answer= -0.827396059946821368141165095479816

となり、10進31桁結果が一致する。

また、総和演算での、整数ﾃｰﾌﾞﾙを持ち、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) とすると、
C=A*B=(-1)**S*2**T*(1.H) では、
S=MOD(S1+S2,2)
T=T1+T2      1≦(1.F)*(1.G)<2
T=T1+T2+1    2≦(1.F)*(1.G)<4
```

mを仮数部のﾋﾞｯﾄ数とすると、2**m*(1.F)=2**m+I (0≦I≦2**m-1)

2**m*(1.G)=2**m+n　(0≦n≦2**m-1)
I、nは正の整数で、浮動小数点ﾃﾞ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≦(1.F)*(1.G)<2は 2**2m≦(2**m+1)*(2**m+n)<2**(2m+1)
　2≦(1.F)*(1.G)<4は 2**(2m+1)≦(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2mﾋﾞｯﾄあるので、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ﾌﾟﾛｸﾞﾗﾑを作成し、確認した。

ﾌﾟﾛｸﾞﾗﾑ

```
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、ﾃｰﾌﾞﾙｻｰﾁや低精度な演算で求めた値を初期値C0とし、反復計算Cn+1=Cn×(2-B×Cn) 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。試作した多倍長ﾗｲﾌﾞﾗﾘでは、1/Bのみの値を計算するﾌﾟﾛｸﾞﾗﾑも多々ある為、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!+.... は |x|<∞ 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|<π/2 となる様にして、I=[|x|/(π/2)] t=|x|-I*π/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、ﾒｯｾｰｼﾞを出す処

理系もある。この様な場合、
|x|/(π/2)、|x|-1*π/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]

$$Sp(z)=\int_0^{z+1} t/(\exp(t)-1)\,dt=\sum_{n=0}^{\infty} Bn\,(-\ln(1-z))^{**}n+1/(n+1)!$$

[5] の右辺の有限個で計算するが、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 Bn, Bn/(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≒10**72) ため、Bnの循環式 [6]

$$\binom{2n+1}{2}\,{}^{*}B1-\binom{2n+1}{4}\,{}^{*}B2+\cdots\cdots+(-1)^{**}(n-1)\binom{2n+1}{2n}\,{}^{*}Bn=n-1/2$$

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

In=1/N Σ (0≦k≦N-1)
f([g1(N)/N*k], [g2(N)/N*k]) [7]
ここで、F1=F2=1 Fn+2=Fn+1+Fn (n=1, 2, ...)
(Fibonacci 数列)
N=Fn g1(N)=1 g2(N)=Fn-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ない。{x} を高精度に計算するには、g2(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ﾃﾞｰﾀ表現には同じ精度の浮動小数点ﾃﾞｰﾀ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
=倍精度+倍精度+倍精度+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=ΣAi (i=1, n)= Σ (i=1, n) Σ j=1, 2) Aij
4倍精度+4倍精度
=Σ (i=1, n) Σ j=1, 4) Aij
倍精度+倍精度+倍精度+倍精度

T=Σ (i=1, n) (Ai*Bi)=Σ (i=1, n)
(Σ (j=1, 2) Σk=1, 2) (Aij*Bik))
4倍精度+4倍精度
=Σ (i=1, n) (Σ (j=1, 4)
Σk=1, 4) (Aij*Bik))
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, 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+.....+1/n (n=1000000)
4倍精度
0.14392726722865723631381127493189O2E+02
4倍精度+4倍精度
0.14392726722865723631381127493189S9E+02
8倍精度
0.14392726722865723631381127493189S9E+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 ﾋﾞｯﾄ まで等しくなっている。

(2) 平方根: 4倍精度-8倍精度
√1+√2+√3+....+√n n=1000000

4倍精度
0.66666716645882210835597876679521O3E+09
4倍精度+4倍精度
0.66666716645882210835597876679519S2E+09
8倍精度
0.66666716645882210835597876679519S2E+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 ﾋﾞｯﾄ 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、配列整数ﾃｰﾌﾞﾙは、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. Paradopoulos, 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(αs) 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

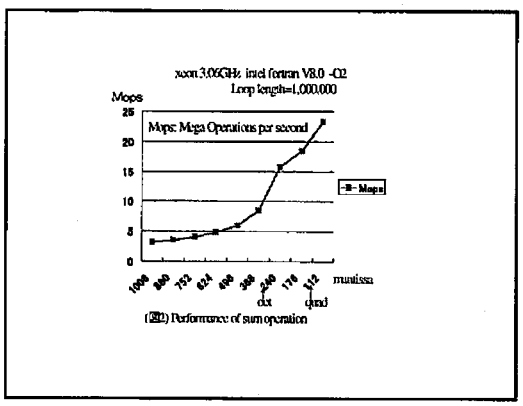
10. 謝辞

精度検証にあたり御指導いただきました 高ｴﾈﾙｷﾞｰ加速器研究機構 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。

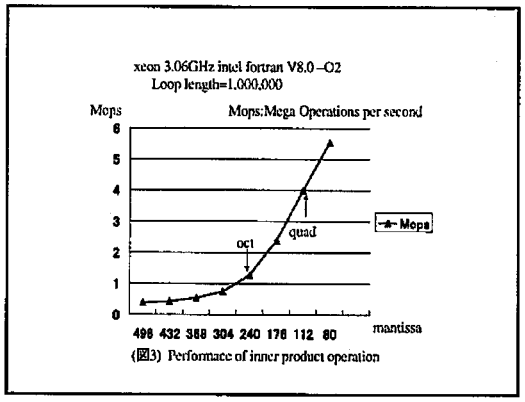

```
a1=(2.0-0.5**51)*2.0**515
a2=(2.0-0.5**50)*2.0**515
b1=(2.0-0.5**51)*2.0**515
b2=(2.0-0.5**50)*2.0**515
s=a1*b1-a2*b2
write(6,1) s
1 format(1h ,2x,'s=',e25.17)
stop
end
```

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

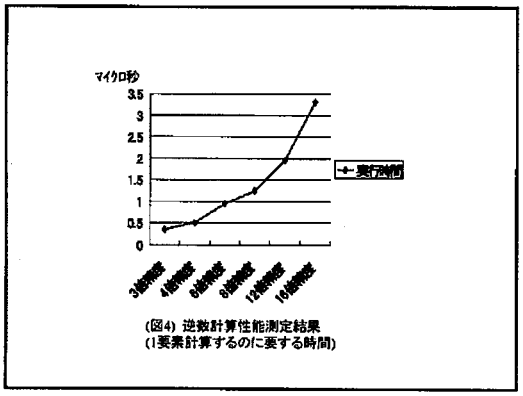
(図1) 引数、結果ともに表現可能な数値で、演算途中でオーバーフローする例



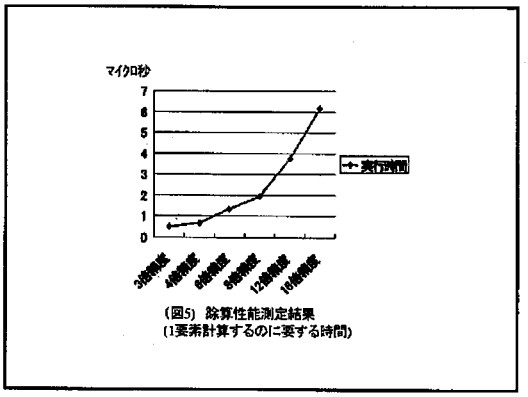
(図2) Performance of sum operation



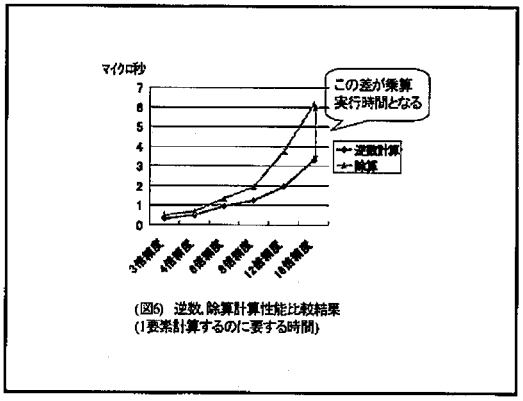
(図3) Performace of inner product operation



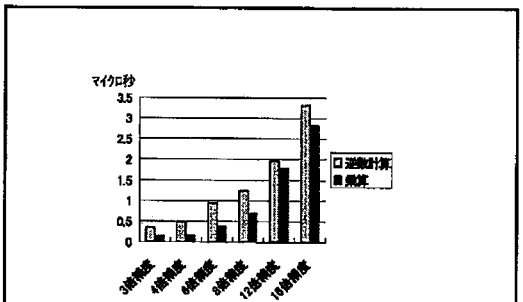
(図4) 逆数計算性能測定結果
(1要素計算するのに要する時間)



(図5) 除算性能測定結果
(1要素計算するのに要する時間)



(図6) 逆数,除算計算性能比較結果
(1要素計算するのに要する時間)



(図7) 逆数,乗算計算性能比較結果
(1要素計算するのに要する時間)