Vol.2015-HPC-152 No.15
2015/12/17

Feynman 4-loop 積分のチューニングとアクセラレータを使用した高速化

濱口信行[†1]　石川正[†1]　湯浅富久子[†1]

概要：Feynman 4-loop 積分（8 次元積分）を精度 10 進 3 桁の結果を得るのに、4 倍精度演算で E5-1660 を用いて約 170000 秒を要した。桁落ちを考慮したチューニングを行い、精度 10 進 11 桁の結果を倍精度演算で得ることができた。
E5-2670、E5-2660、Phi5110P を用いて高速化を実施するとともに Pezy-SC に移植して、数秒にまで計算時間を短縮できた。本発表では、その性能および精度向上の経過内容を詳細に説明する。

Improvement of Accuracy and performance with accelerator on Feynman four loop massless integral

Nobuyuki Hamaguchi[†1]　Tadashi Ishikawa[†1]　Fukuko Yuasa[†1]

1. はじめに

Feynman ループ積分の直接計算法[1]は多次元数値積分に帰着されます。一般に多次元数値積分では、次元数が大きくなると、決定論的計算法では演算量が膨大となり(多次元の呪い)、統計的計算法では、高精度の結果を得るのが困難になる為[2]、実行時間と結果の精度を両立させるのが困難になる。今まで扱った Feynman ループ積分は、6 次元以下の多次元数値積分に帰着されたので、二重指数関数型(DE)積分法でアクセラレータを効率良く使用でき、短い実行時間で高精度の結果を得る事が出来た。[3]
しかし更に複雑な 4-loop 積分を扱うと、その演算量、結果の精度、効率的なアクセラレータの使用に関する関係が従来と大きく異なったものとなっている。本報告では新たに発生した問題への対策を紹介します。

尚今回使用した主要な計算機とそのカタログ性能は以下のものです。
Phi5110P:1011GFLOPs、E5-2670:333GFLOPs、SR16000:980GFLOPs、E5-2660:282GFLOPs、Pezy-SC :1500GFLOPs です。

2. 4-loop 積分[4]

4-loop 積分は $\mathrm{I}=\int_\Omega \frac{1}{\mathrm{C}(\mathrm{D}-\mathrm{i}\varepsilon\mathrm{C})}\mathrm{d}\Omega$

$$\begin{aligned}\Omega=\{(&x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9)\mid\\ &x_1\ge 0, x_2\ge 0, x_3\ge 0, x_4\ge 0,\\ &x_5\ge 0, x_6\ge 0, x_7\ge 0, x_8\ge 0, x_9\ge 0,\\ &x_1+x_2+x_3+x_4+x_5+x_6+x_7+x_8+x_9=1\}\end{aligned}$$

$x_9=1-x_1-x_2-x_3-x_4-x_5-x_6-x_7-x_8$より
積分としては8次元積分となります。
微小量$i\varepsilon$とDEでのεを区別するため、微小量のεをepsと記述します。また変数の表現で
$x_{i,j,k,....}=x_i+x_j+x_k+.....$と表しています。

今回扱った問題は、先行研究[5]の自己エネルギー型 Feynman 4-loop 積分の 2 つの計算事例であり、それぞれ C、D が与えられる。それぞれの解析解は、

M_{44}は$\frac{441}{8}\zeta_7+\mathrm{O(eps)}$で$\mathrm{O(eps)}=0$、
M_{45}は$36{\zeta_3}^2+\mathrm{eps}\times(108\zeta_3\zeta_4-378\zeta_7)+\mathrm{O(eps^2)}$で
$\mathrm{eps}=0$、$\mathrm{O(eps^2)}=0$の場合としている。
今回はM_{44}に対してチューニングを行い、
そこで得た知見をM_{45}に適用している。

2.1 M_{44}

C、D および解析解、E5-1660(3.3GHz) 1cpu で 4 倍精度演算の実行時間、実行結果は以下の通りである。

†1 高エネルギー加速器研究機構

$$C = x_{167}x_{278}x_{349}x_{589} - x_{349}x_{589}x_7^{\ 2} - x_{167}x_{349}x_8^{\ 2} - x_{167}x_{278}x_9^{\ 2} + x_7^{\ 2}x_9^{\ 2}$$

$$\begin{aligned} D = &-x_{167}x_{278}x_3^{\ 2}x_{589} - x_{167}x_2^{\ 2}x_{349}x_{589} - x_1^{\ 2}x_{278}x_{349}x_{589} \\ &+ x_{123}x_{167}x_{278}x_{349}x_{589} - 2x_1x_2x_{349}x_{589}x_7 + x_3^{\ 2}x_{589}x_7^{\ 2} \\ &- x_{123}x_{349}x_{589}x_7^{\ 2} + x_{167}x_3^{\ 2}x_8^{\ 2} + x_1^{\ 2}x_{349}x_8^{\ 2} \\ &- x_{123}x_{167}x_{349}x_8^{\ 2} - 2x_{167}x_2x_3x_8x_9 - 2x_1x_3x_7x_8x_9 \\ &+ x_{167}x_2^{\ 2}x_9^{\ 2} + x_1^{\ 2}x_{278}x_9^{\ 2} - x_{123}x_{167}x_{278}x_9^{\ 2} \\ &+ 2x_1x_2x_7x_9^{\ 2} + x_{123}x_7^{\ 2}x_9^{\ 2} \end{aligned}$$

解析解 $= \dfrac{441\zeta_7}{8} = 55.5852539156784$

$eps = 0.46291254 \times 10^{-6}$

実行時間＝177427秒

実行結果＝55.5900070293401

相対誤差8.55×10^{-5}

2.2 M_{45}

C、D および解析解、E5-1660(3.3GHz) 1cpu で 4 倍精度演算の実行時間、実行結果は以下の通りである。

$$\begin{aligned} C = &-x_{279}x_{349}x_5^{\ 2} + x_{1567}x_{279}x_{349}x_{589} - x_{349}x_{589}x_7^{\ 2} \\ &-2x_{349}x_5x_7x_9 - x_{1567}x_{279}x_9^{\ 2} - x_{1567}x_{349}x_9^{\ 2} \\ &+x_5^{\ 2}x_9^{\ 2} - x_{1567}x_{589}x_9^{\ 2} + 2x_5x_7x_9^{\ 2} \\ &+x_7^{\ 2}x_9^{\ 2} + 2x_{1567}x_9^{\ 3} \end{aligned}$$

$$\begin{aligned} D = &\ x_{279}x_3^{\ 2}x_5^{\ 2} + x_2^{\ 2}x_{349}x_5^{\ 2} - x_{123}x_{279}x_{349}x_5^{\ 2} \\ &- x_{1567}x_{279}x_3^{\ 2}x_{589} - x_{1567}x_2^{\ 2}x_{349}x_{589} - x_1^{\ 2}x_{279}x_{349}x_{589} \\ &+ x_{123}x_{1567}x_{279}x_{349}x_{589} - 2x_1x_2x_{349}x_{589}x_7 + x_3^{\ 2}x_{589}x_7^{\ 2} \\ &- x_{123}x_{349}x_{589}x_7^{\ 2} + 2x_1x_{279}x_3x_5x_9 - 2x_1x_2x_{349}x_5x_9 \\ &+ 2x_2x_3x_5^{\ 2}x_9 - 2x_{1567}x_2x_3x_{589}x_9 + 2x_2x_3x_5x_7x_9 \\ &+ 2x_3^{\ 2}x_5x_7x_9 - 2x_{123}x_{349}x_5x_7x_9 - 2x_1x_3x_{589}x_7x_9 \\ &+ x_{1567}x_2^{\ 2}x_9^{\ 2} + x_1^{\ 2}x_{279}x_9^{\ 2} - x_{123}x_{1567}x_{279}x_9^{\ 2} \\ &+ 2x_{1567}x_2x_3x_9^{\ 2} + x_{1567}x_3^{\ 2}x_9^{\ 2} + x_1^{\ 2}x_{349}x_9^{\ 2} \\ &- x_{123}x_{1567}x_{349}x_9^{\ 2} + 2x_1x_2x_5x_9^{\ 2} - 2x_1x_3x_5x_9^{\ 2} \\ &+ x_{123}x_5^{\ 2}x_9^{\ 2} + x_1^{\ 2}x_{589}x_9^{\ 2} - x_{123}x_{1567}x_{589}x_9^{\ 2} \\ &+ 2x_1x_2x_7x_9^{\ 2} + 2x_1x_3x_7x_9^{\ 2} + 2x_{123}x_5x_7x_9^{\ 2} \\ &+ x_{123}x_7^{\ 2}x_9^{\ 2} - 2x_1^{\ 2}x_9^{\ 3} + 2x_{123}x_{1567}x_9^{\ 3} \end{aligned}$$

解析解$= 36\zeta_3^{\ 2} = 52.017868743610$

$eps = 0$

実行結果　52.024759995682

実行時間335300秒

相対誤差$= 1.32 \times 10^{-4}$

3. 二重指数関数型積分法[6]

多次元数値積分で被積分関数が端点特異点を持つ場合二重指数関数型積分法が演算量、精度の面から最適な積分法である。今回扱った Feynman ループ積分は原点で発散し、有限な積分値が存在しているケースである。

使用した変数変換は$x = \phi(t) = \dfrac{1}{2}\tanh(\dfrac{\pi}{2}\sinh(t)) + \dfrac{1}{2}$

$$\phi'(t) = \frac{\frac{\pi}{4}\cosh(t)}{\cosh^2(\frac{\pi}{2}\sinh(t))} = x(1-x)\sqrt{(\log(\frac{x}{1-x}))^2 + \pi^2}$$

有限項での打ち切り$\phi(t_1) = \varepsilon_1$、$\phi(t_2) = 1 - \varepsilon_2 \quad (0 < \varepsilon_1, \varepsilon_2 << 1)$

とすると近似的に$\sinh(-t_1) = \dfrac{1}{\pi}\log(\dfrac{1}{\varepsilon_1})$、$\sinh(t_2) = \dfrac{1}{\pi}\log(\dfrac{1}{\varepsilon_2})$

となる。

$x = p(0 < p < 1)$の場合の$|f(x)|\phi'(t)$の値をJ_pとする。積分値の精度を見積もる場合、$0 < \varepsilon << 1$に対し$J_\varepsilon / J_{0.5}$、$J_{1-\varepsilon}/J_{0.5}$の値を見る。この値の大きい方が積分計算結果の精度の指標となる。以下にいつくかの例を示す。

例えばベータ関数$\int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx \quad (0 < \alpha、\beta)$では

$$f(x) = x^{\alpha-1}(1-x)^{\beta-1}、|f(x)|\phi'(t) = x^\alpha(1-x)^\beta\sqrt{(\log(\frac{x}{1-x}))^2 + \pi^2}$$

$$J_{0.5} = (\frac{1}{2})^{\alpha+\beta}\pi、J_\varepsilon \fallingdotseq \varepsilon^\alpha\sqrt{(\log(\varepsilon))^2 + \pi^2}、J_{1-\varepsilon} \fallingdotseq \varepsilon^\beta\sqrt{(\log(\varepsilon))^2 + \pi^2}$$

$\alpha = 0.01$、$\beta = 1.01$の場合、$\varepsilon = 10^{-300}$で$J_\varepsilon / J_{0.5} = 0.45$、$\varepsilon = 10^{-16}$で$J_{1-\varepsilon}/J_{0.5} = 1.65 \times 10^{-15}$となる。すなわち指数部15ビットの演算が必要になる。これは$\int_0^1\int_0^{1-x}\dfrac{1}{(xy)^{1-\eta}}dydx = \dfrac{1}{\eta}\int_0^1 x^{\eta-1}(1-x)^\eta dx \quad (\eta = 0.01)$で精度の良い結果を得るには、拡張倍精度、IEEE754－2008形式の4倍精度演算が必要であった事[7]を裏付けている。

また$\alpha = 1.01$、$\beta = 0.1$の場合、IEEE754－2008の4倍精度演算で$\varepsilon = 10^{-30}$で$J_{1-\varepsilon}/J_{0.5} = 0.048$となる。1により近い数値表現ができる2つの倍精度変数の和で表す4倍精度演算が必要になる事を示している。

精度改善例として、$\alpha = \beta = 0.01$の場合を考える。この場合、変数を指数部15ビットの変数の和で表す演算が必要となる。

解析解=199.967577315886

倍精度演算結果=127.642554377576

4 倍精度演算結果=154.2730534625816

2 つの 4 倍精度変数の和(DQ:8 倍精度)の演算結果

199.967577315886　(解析解と一致)

一方$\int_0^1 x^{\alpha}(1-x)^{\beta}dx$という形の積分は、前の式の

$\alpha=\beta=1.01$の場合に相当する。この場合$\varepsilon=10^{-12}$と

しても$J_{\varepsilon}/J_{0.5}=J_{1-\varepsilon}/J_{0.5}\fallingdotseq 3\times10^{-11}$となり、

倍精度演算で十分な精度の結果が得られる事　が

示されている。

$$\int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx=\frac{(\alpha+\beta+1)(\alpha+\beta)}{\alpha\beta}\int_0^1 x^{\alpha}(1-x)^{\beta}dx$$

で倍精度演算で計算すると実行結果は

199.967577315886　(解析解と一致)

となっている。

4. 精度解析とチューニング

4.1 精度解析

M_{44},M_{45}の解析解にはツエータ関数ζ_7、ζ_3[8]が

含まれている。

$$\zeta_3=\frac{1}{2}\int_0^1\frac{(\log(x))^2}{1-x}dx、\zeta_7=\frac{1}{720}\int_0^1\frac{(\log(x))^6}{1-x}dx$$

問題の積分は多次元積分であるが、

最終的に$\int_0^1\frac{(\log(x))^{2n}}{1-x}dx\ (n=1,3)$の

積分となる事が想定される。

$f(x)=\frac{(\log(x))^{2n}}{1-x}$から

$$|f(x)|\phi'(t)=x(\log(x))^{2n}\sqrt{(\log(\frac{x}{1-x}))^2+\pi^2}$$

$$J_{0.5}=\frac{\pi}{2}(\log(2))^{2n}\fallingdotseq 0.5\sim1\quad(n=1,3)$$

$x=1-\varepsilon$なら$\log(x)\fallingdotseq-\varepsilon$、$|f(x)|\phi'(t)\fallingdotseq\varepsilon^{2n+1}\log(\frac{1}{\varepsilon})$

$\varepsilon=10^{-5}$で$J_{1-\varepsilon}/J_{0.5}$は十分小さくなる。

$x=\varepsilon$なら$|f(x)|\phi'(t)\fallingdotseq\varepsilon(\log(\frac{1}{\varepsilon}))^{2n+1}$

$\varepsilon=10^{-30}$なら$J_{\varepsilon}/J_{0.5}$は十分小さくなるので精度の

良い積分値を得るためには指数部は11ビットで

十分である事がわかる。演算中の桁落ちは

$x=1-\varepsilon$で$1-x$の演算時が最も大きくなる。

$x=1-\varepsilon$に関する変数はx_9でM_{44}にはx_9の2次式、

M_{45}にはx_9の3次式が現れるので実行に際し桁落ちの

影響が小さいM_{44}のチューニングを先に実施した。

4.2 チューニング

4.2.1 チューニング方法

変数変換区間$[\varepsilon,1-\varepsilon]$の二重指数関数型積分法での

精度改善方法には以下の2つの方法があるが、

それぞれに問題がある。

(1)εを固定し、刻み幅hを小さく(分点数Nを大きく)する。

従来はこの方式で行っていたがM_{44}では演算量がNの

8乗に比例するため膨大となる。

(2)分点数を一定にしてεを大きく(刻み幅hを小さく)する。

精度が改善されるかどうかが不明。

M_{44}の最初の実行結果が解析解より大きいので、

(2)の方法で解析解より小さいhを求めて、hの採る値の範囲

を狭めて、精度の良い結果を得る方法を実施した。

M_{45}も最初の実行結果が解析解より大きいので同じ

手法が使える事になる。

4.2.2 チューニング手順

M_{44}では精度の良い積分値を得るには、指数部15ビット

は必須ではないので､4倍精度演算を倍精度演算にして

実行したが桁落ちのため異常終了した。原因は設定

していた$\varepsilon=2.14\times10^{-14}$で倍精度演算のマシンイプシロン

$\delta=2^{-53}$に対し、$\varepsilon^2<\delta<\varepsilon$であった事による。

$\varepsilon^2>\delta$になる様に$h=0.2$、$N=25$、$\varepsilon=3.48\times10^{-8}$、

$eps=0$にして実行し、以下の結果を得た。

55.5851270443804

相対誤差2.28×10^{-6}で4倍精度演算より精度が良くなって

いる。変数変換区間$[\varepsilon,1-\varepsilon]$　$(0<\varepsilon<<1)$で

$h=\frac{A}{N-1}$、$t_{max}=\frac{A}{2}$、$t_{min}=-t_{max}$とすると,今回は

$h=0.2$、$A=4.8$、$t_{min}=-2.4$で実行した事になる。

$\varepsilon^2=\delta$、$N=25$とすると、Aの値は4.932。今回の実行結果は

解析解より小さいのでAを4.8から4.932の間でより精度の

良い結果を求めた。

$A_i=4.8+0.01\times i$で計算し、

A_iでの実行結果＜解析解＜A_{i+1}での実行結果

となるiを求める。

次に$A_j=A_i+0.001\times j$として解析値に対するAの範囲

を狭めていく。

最終的には倍精度演算で

A=4.8133　　　result=55.5852536971447

A=4.81332　　　result=55.5852538793107

A=4.813324　　　result=55.5852539157413

と 10 進 11 桁の精度の結果を得た。

M_{45}はx_9の3次式を含むため、M_{44}より桁落ちの影響が大きいが、倍精度変数を2つつなげた4倍精度演算(有効ビット数106ビット)ほどのビット数は必要なく、拡張倍精度演算(有効ビット数65ビット)でM_{44}と全く同じ方法で精度が向上した。分点数NもM_{44}と同じくN＝25とした。

解析解　　　　　　= 52.017868743610
A=4.785582　result=52.017868748703
A=4.7855815　result=52.017868743216
A=4.78558154 result=52.017868743634
と 10 進 12 桁の精度の結果を得た。

5. 性能測定結果

M_{44}、M_{45}ともにN＝25、最終精度を得た条件で実行している。

5.1 M_{44}性能測定結果

精度チューニングのみ行ったプログラムで実行。
最適化オプションの影響を受けやすく、
-01 -openmp （-mmic）で実施している。
使用言語は C++。

表1	倍精度演算実行結果一覧	
CPU	smp	実行時間(秒)
E5-2670	16	227.7403
Phi5110P	240	1455.5575
E5-2660	16	267.3872
E5-2660	32	189.4230

E5-1660 で実行に最初 2 日かかっていたものが E5-2670 で 4 分弱で実行出来ている。
カタログ性能比では Phi5110P は E5-2670 の 3 倍の性能を有しているが実行結果では 6 倍以上遅くなっている。
E5-2660 の smp 数 16、32 はハイパースレッドを使用するかどうかで分点数が N=25 のため smp 数 32 の場合 smp 数 16 の場合の約 1.5 倍の性能向上となっている。

5.2 M_{45}性能測定結果

精度チューニングのみ行ったプログラムで実行。
実行は言語 c++　で拡張倍精度演算と言語 FORTRAN で 4 倍精度演算を行っている。
拡張倍精度演算では最適化オプションの影響を受けやすく、-01 -openmp （-mmic）で実施している。
4 倍精度演算では最適化オプションの影響を受けないため、標準オプションで実施している。

表2	拡張倍精度実行結果	
CPU	smp	実行時間(秒)
E5-2670	16	688.6699
Phi5110P	240	3379.0798
E5-2660	16	900.5380
E5-2660	32	600.1981

E5-1660 で実行に最初 4 日かかっていたものが E5-2670 で 11 分強で実行出来ている。拡張倍精度演算のため Phi5110P は E5-2670 の 3/4 程度の性能だが実行結果はそれ以上に差がある。
これは分点数が N=25 と小さい事による。

E5-2660 の smp 数 16、32 はハイパースレッドを使用するかどうかで分点数が N=25 のため smp 数 32 の場合 smp 数 16 の場合の約 1.5 倍の性能向上となっている。
この関係は倍精度演算､拡張倍精度演算で変わりはない。

4 倍精度演算では以下の様になっている。

表3	4倍精度実行時間	
CPU	smp	実行時間(秒)
E5-2670	16	30367.7276
SR16000	64	6387.4925

E5-2670 と SR16000 での倍精度演算でのカタログ性能比は約 1:3。性能比で正規化して 4 倍精度の性能は SR16000 が E5-2670 の約 1.6 倍。IEEE754-2008 形式の 4 倍精度と 2 つの倍精度変数の和の 4 倍精度演算では従来 2～2.5 倍の差があったが昨今はその差が縮まっている。
ただ SR16000 の 4 倍精度の実行時間は E5-2670、E5-2660 の 10 倍以上遅い､拡張倍精度の性能が出にくいとされる Phi5110P の 2 倍弱かかっているので､拡張倍精度演算ををサポートしていない SR16000 はこの種の計算には不向きとも言える。

6. Phi5110P への適応性

倍精度演算では従来の 6 次元積分ではカタログ性能に近く､Phi5110P は E5-2670 の 2-3 倍の性能が出ていた。[3]

そこで新たな6次元積分Laporta I[9]とM_{44}での比較を行った。

Laporta *I* (6次元)

$$I=\int_{\Omega}\frac{C}{D^3}d\Omega$$

$$\Omega=\{(x_1,x_2,x_3,x_4,x_5,x_6,x_7)\mid x_1\ge 0,x_2\ge 0,x_3\ge 0,x_4\ge 0, x_5\ge 0,x_6\ge 0,x_7\ge 0, x_1+x_2+x_3+x_4+x_5+x_6+x_7=1\}$$

$$d\Omega=dx_6dx_5dx_4dx_3dx_2dx_1$$

$$x_7=1-x_1-x_2-x_3-x_4-x_5-x_6$$

$$C=(x_1+x_2+x_3+x_4+x_5)(x_1+x_2+x_3+x_6+x_7)-(x_1+x_2+x_3)^2$$

$$cc=x_1m_1^2+x_2m_2^2+x_3m_3^2+x_4m_4^2+x_5m_5^2+x_6m_6^2+x_7m_7^2$$

$$\begin{aligned}D=&-C\times cc+s(x_1x_2(x_4+x_5+x_6+x_7)\\&+x_1x_5x_6+x_2x_4x_7-x_3x_4x_6)\\&+tx_3(-x_4x_6+x_5x_7)\\&+p_1^2(x_1x_3(x_4+x_5+x_6+x_7)+x_3x_4(x_6+x_7))\\&+p_2^2(x_2x_3(x_4+x_5+x_6+x_7)+x_3x_6(x_4+x_5))\\&+p_3^2(x_4x_5(x_1+x_2+x_3+x_6+x_7)+x_4x_6(x_2+x_3)\\&\quad+x_1x_5x_7)\\&+p_4^2(x_6x_7(x_1+x_2+x_3+x_4+x_5)+x_4x_6(x_1+x_3)\\&\quad+x_2x_5x_7)\end{aligned}$$

$$m_1^2=m_2^2=m_3^2=m_4^2=m_5^2=m_6^2=m_7^2=1$$

$$s=t=1$$

$$p_1^2=p_2^2=p_3^2=p_4^2=1$$

解析解 = 0.0853513981538

6.1 Laporta I の実行結果

N=64、N=96、N=128 での実行結果は以下のとうり。

解析解	0.0853513981538
N=64	0.0853513981538937400
N=96	0.0853513981538817218
N=128	0.0853513981538628203

表4	Laporta I 性能測定結果	
	実行時間一覧表（秒）	
N	Phi5110P	E5-2670
64	21	57
96	252	642
128	1516	3653

プログラムでは積分領域Ωから$[0,1]^6$に変換して
その後3重ループを一重化し,2重ループとして実行。
ループ長はそれぞれ64^3,96^3,128^3となる。また
問題は最適化の影響は受けないため標準オプション
で実行している。

ソース上からの演算量は$150\times N^6$ *FLOP*でこの場合
の性能は以下の様になっている。

表5 性能一覧表(GFLOPs)		
N	Phi5110P	E5-2670
64	491	181
96	466	183
128	435	181

実行効率は Phi5110P 45%、E5-2670 55%となっている。

6.2 M_{44}

8次元の場合積分領域を$[0,1]^8$とすると、Jacobianが変数の
28乗となるため、二重指数関数型積分の変数変換区間
$[\varepsilon,1-\varepsilon]$のεの大きさに、倍精度変数の表現可能な数値範囲の
制限から、今回のεの値が取れなくなる。$\varepsilon=10^{-6}$とすると
$\varepsilon^{28}=10^{-168}$でこの値の乗算ができなくなる。
このため、多重ループの一重化では、外側4重ループを
変数の値(x,y,z,u)と重み係数、区間係数(区間[0,1]から変数の値
を求める係数)を先に求めておき、その後5重ループを計算
するようにする。この場合、εの値を変更する必要がない。
この時の結果は E5-2670、Phi5110P とも
result= 55.5852539157264 で一重ループ化する前
と精度の変化はない。
測定は Intel Fortran Compiler XE V13.0 を用いた。

表6 E5-2670,Phi5110P実行結果	
E5-2670 16smp	
チューニング	実行時間(秒)
多重DOループの一重化	161.7189
Phi5110P 240smp	
チューニング 多重DOループの一重化	実行時間 (秒)
60smp	641.7513
120smp	386.7077
180smp	320.6218
240smp	261.8692

E5-2670 と Phi5110P の性能差は 6 倍強から
1.61 倍と縮まっては来ているがまだカタログ
性能比には遠く及ばない。
最適化オプションは依然-01 のため最適化効果
の影響が Phi5110P に大きく出ている事による。
ただ最適化を強化したオプションで実行すると、
0<D<<1 の値が D=0 となりゼロ割が発生する。
0<D<<1 の D は結果にほとんど影響しないので、
$eps = 10^{-12}$を入れ、共通項をまとめる最適化を
止めるように各項を　括弧でくくり、
標準オプション(–O2)で実行するように変更した。

実行結果は、
result=　 55.5852539155209
で精度的には問題のない結果が得られた。
性能測定結果は以下の様になっている。

表7 標準オプション適用効果

$eps = 10^{-12}$

CPU	実行時間(秒)
E5-2670 16smp	130.0995
Phi5110P 240smp	56.6998

ソース上からの演算量は$150 \times N^6$FLOPとなり、$N = 25$
では20447GFLOPとなり、E5 – 2670で157GFLOPs、
実行効率47%、Phi5110Pで361GFLOPs、実行効率36%
とNの大きさを考えるとLaporta　Iの6次元積分の結果と
遜色ない値となっている。

アクセラレータで効率よく性能を出すには、一般的には
(1) ループ長を長くすること
(2)最適化を効かせること
が必要であり、Phi5110P では 60core、4SIMD/1core が効
率よく計算出来る様にする事と言える。

7. Pezy-SC への移植

実行環境は 1024PE、8 スレッド SMT で最大スレッド数は
8192 スレッド。サポートしている演算は単精度演算と倍
精度演算で使用言語は OpenCL である。

実行したプログラムはM_{44}でhost(E5 – 2660)側でDEの分点
と重み係数テーブルの作成、およびPezy – SCの結果の
総計を求める計算、Pezy – SCで積分値の部分和を計算して
いる。host側の演算負荷は非常に小さく、実行時間的には
無視できる。

7.1 プログラムへの制限事項

7.1.1 スレッド数

指定できるスレッド数は 128 の倍数という制限がある。
DE の分点数 N には、多重ループの一重化をしない場合 128
の倍数、二重ループ、三重ループ、四重ループの一重化
をする場合、それぞれ 16 の倍数、8 の倍数、4 の倍数と
いう制限がつく。

7.1.2 テーブルの個数と演算

多重ループを外側ループと内側ループに分け外側ルー
プを host 側、内側ループを Pezy-SC 側で実行する。
積分領域を変更しない場合、外側ループの一重化で、
二重ループ、三重ループ、四重ループの一重化をする場
合、host 側で作成するテーブルの個数はそれぞれ 6、7、
8 となりメモリ負荷の問題が生じる。

積分領域を$[0,1]^8$に変換する場合、*host*側で作成する
配列は最小2個で、内側ループのテーブル参照方式で
以下の3つの場合がある。

(1) host 側で作成するテーブル 2 個、Pezy-SC 平方根で、
べき乗根の計算なし。
(2) host 側で作成するテーブル 2 個、Pezy-SC で平方根、
べき乗根の計算あり。
(3) host 側で作成するテーブル 2 個、Pezy-SC で平方根、
べき乗根の計算あり。

7.1.3 基本演算

この積分計算では、必ず浮動小数点除算が必須で、内側
ループのテーブル参照方式で Pezy-SC で平方根および立
方根などのべき乗根計算が必要になる事がある。
これに関係する Pezy-SC の演算器構成として、除算器が
16PE で共有、平方根演算器が 16PE で共有、単精度しか
ない事がある。
Pezy-SC で倍精度演算で配列サイズ 65536 で c=a*b の演
算を 65536 回実行した場合の実行時間 0.092508 秒で
46GFLOPs に対し、除算は 1.600070 秒、平方根計算
4.832607 秒、立方根計算 5.6879671 秒となっている。
乗算の測定は、1 演算に対し、16 バイトのデータ参照と
8 バイトのデータ格納がある。乗算の性能がカタログ性
能 750GFLOPs とすると、演算量 4.295GFLOP から実行時間
0.092508 秒、演算時間　0.005727 秒、データ参照、格納
時間は 0.086781 秒となる。このため、同じループ内にあ
る乗算に対する除算、平方根計算、立方根計算時間比は
この測定の倍率以上に大きくなる。このため、平方根計
算や立方根計算が必要となるテーブル参照方式は避ける
必要がある。

7.2 並列化チューニング

7.2.1 メモリ負荷削減

積分領域を$[0,1]^8$に変換して使用する配列を2つにする。
この場合、変数変換区間$[\varepsilon, 1-\varepsilon]$の$\varepsilon$の値を以下の条件を満たす様に変更した。
Jacobianでε^{28}、被積分関数からε^{25}の項があるので
$\varepsilon^{53} \geq 2^{-1023}$から$\varepsilon \geq 1.547 \times 10^{-6}$、$t_{min} \geq -2.1556$。

7.2.2 分点数

E5−2670、Phi5110PではN = 25、相対誤差 = 1.13×10^{-12}。
N = 32とすると演算量は$(\frac{32}{25})^8 = 7.2$倍に増加する。
N = 16とすると精度は$\exp(-\frac{25}{\log(25)} + \frac{16}{\log(16)}) = 7.36$倍悪くなる。これより、Nの値としては$16 \leq N \leq 32$の範囲で選択する必要がある。

実際に選択できるNの値は多重ループを一重化する場合、二重ループの一重化ではN = 16,32、三重ループの一重化ではN = 16,24,32、四重ループの一重化ではN = 16,20,24,28,32となる。

7.2.3 ループ構成

外側ループを一重化する場合、二重ループだとループ長が 1024 以下、三重ループだと、内側ループの計算で立方根の計算が必要になるので、外側ループ、内側ループともに 4 重ループとし外側ループは一重化し、内側ループは整数演算を少なくするため 4 重ループにしている。

7.2.4 変数変換区間の調整

変数変換区間$[\varepsilon, 1-\varepsilon]$を$[\varepsilon_1, 1-\varepsilon_2]$ $(\varepsilon_1 \neq \varepsilon_2)$にする方法には以下の方法がある。
(1)外側ループ、内側ループを同時に行う(区間調整)。
(2)内側ループの分点数を外側ループの分点数の約数にする。
(2)の方法では、外側ループ、内側ループのそれぞれのε_1で
$\varepsilon_{外側ループ} = \varepsilon_1$、$\varepsilon_{内側ループ} = \varepsilon_1$とし、
$\mathrm{Jacobian} = \varepsilon^{28} = \varepsilon_{外側ループ}^{22} \times \varepsilon_{内側ループ}^{6}$
$\varepsilon_{外側ループ} < \varepsilon < \varepsilon_{内側ループ}$として精度を上げる事が出来る。
この場合のεの値を$<\varepsilon>$で表す。

7.2.5 イプシロン算法の応用

今回の様に端点特異点のみ(eps = 0で実行可能)で0近傍の$x = \varepsilon$の値の制限がある積分では、イプシロン算法で精度を向上させる事が出来る[7]。

このとき使用した値は　$eps = 1.2^{-75} \sim 1.2^{-89} = 1.15 \times 10^{-6} \sim 8.97 \times 10^{-8}$から$eps = 10^{-6} \sim 10^{-7}$の一点のepsで精度の良い結果を求める様にしている。

7.3 実測結果

7.3.1 測定ケースと条件詳細

今回表 8、表 9 に示す条件で 5 つのケースで測定した。
もっとも自然な方法と言えるのが case1 である。

表8 測定ケースの分点数

ケース	外側ループ	内側ループ
case1	16	16
case2	32	8
case3	24	12
case4	20	10
case5	28	7

表9 測定諸条件

ケース	区間調整	＜ε＞	eps
case1	無	1.56E−06	5.1160408E−07
case2	無	1.78E−06	1.03805E−06
case3	有	6.12E−06	2.0755E−07
case4	有	2.53E−06	1.0E−06
case5	有	5.95E−06	1.0E−07

7.3.2 演算結果

結果は以下の様になっていて十分な精度となっている。

解析解　55.5852539156784
case1　55.585253915673604
case2　55.585253915795754
case3　55.585253915060420
case4　55.585253915256686
case5　55.585253915764497

7.3.3 実行時間と性能

ソース上からの演算量　は外側ループの分点数を　N_1、内側ループの分点数を　N_2とすると$198 \times N_1^4 \times N_2^4$となる。

5 ケースの実行時間と実行性能は以下の様になっている。

表10 性能測定結果一覧表		
ケース	実行時間 (秒)	実行性能 (GFLOPs)
case1	3.222556	264
case2	3.256121	261
case3	5.149891	264
case4	1.215714	261
case5	1.134380	257

7.3.4 倍精度除算の影響

case1 では実行性能 264GFLOPs、実行効率 17.6%で、この実行効率は除算の影響が大きいと考えられる。
その影響を見るために演算結果を考慮せず、
case1 の除算の出るソース部分(必須で避けられない)
sum+=jacobi*d*w1*w2/(cc*(d*d+eps*eps*cc*cc));
を sum+=jacobi*d*w1*w2*(cc*(d*d+eps*eps*cc*cc));
(演算量は 850.4GFLOP と変更なし)
sum+=jacobi*d*w1*w2;
(演算量は 850.4GFLOP から 820.3GFLOPs となる)
と変更して測定した。その結果はそれぞれ、
1.147540 秒　741GFLOPs、実行効率 49.4%
0.969672 秒　846GFLOPs、実行効率 56.4%
となっている。
これは除算器の強化が有効である事を示している。

8. まとめ

今回の精度、性能改善から以下の事が言えます。

(1) 最初の粗い精度解析が有効で重要である。
(2) 二重指数関数型積分法では分点数をあまり大きく取る必要がない。
(3) 分点数を一定にして変数変換区間を変更させる精度チューニングは端点特異点のみを持つ積分に対し、非常に有効で同時に性能チューニングにもなる。
(4) 微小量ｉε を付加して計算する方法は eps=0 で精度不足の場合に効果があるのみならず、最適化の弊害を防ぐ事にも適用できる。

また拡張倍精度演算での結果から､アクセラレータで指数部 15 ビット演算を倍精度演算と同程度に強化をすれば適用範囲が大きく広がると言える。

9. おわりに

8次元積分に帰着される 4-loop 積分計算で従来使用していた二重指数関数型積分の精度解析を詳細に行い、新たに分点数一定で変数変換区間幅の変更によるチューニング、分点数を複数個持つチューニング、イプシロン算法の応用により、アクセラレータで精度よく、多次元の呪いが克服できた。今後はまだチューニングを実施していない 3-loop 積分（7 次元積分に帰着される）への適用およびより精度の厳しい積分にアクセラレータを使用した拡張倍精度、4 倍精度以上の多倍長演算を高速化していく予定です。尚、より詳細な結果や別分野の結果に関しては、高性能計算の扉[10]に記載しています。

謝辞　本研究は JSPS 科研費 15H03668 の助成を受け、HPCI 戦略プログラム分野 5「物質と宇宙の起源と構造｣および株式会社 ExaScaler との共同研究の元で実施したものです。

参考文献

1)　高エネルギー素粒子反応に対する高次補正を含む自動計算プログラム（その２）日本物理学会　第６６回年次大会
高エネルギー加速器研究機構:湯浅富久子、石川正、栗原良将、清水韶光、濱口信行　工学院大学院大学:加藤潔
2)　電子計算機のための数値計算法　II
山内二郎、森口繁一、一松信　1981 年 培風館
3)　ファインマンループ積分によるアクセラレータの精度、性能評価
濱口信行、石川正　(高エネルギー加速器研究機構)
情報処理学会研究報告 IPSJ SIG　Technical report
2014-HPC-147(8) 2014/12/9
4)　完全数値的手法によるマルチループ積分の計算手法
湯浅富久子、石川正、加藤潔、台坂博、中里直人
日本物理学会 2015 年秋季大会
5)　FourLoop Massless Propagators:anAlgebraic Evaluation of ALL Master Integrals、Nuclear Physics
B837(2010) 186-220 P.A.Bailkov and K.G.Chetrykin
6)　数値解析　森正武 2002 年　共立出版
7)　SR11000 での 4 倍精度、多倍長精度演算使用例
濱口信行（(株)日立製作所ソフトウエア事業部)
九州大学情報基盤研究開発センター　全国共同利用システム広報　Vol2 No.3 p.p.102-108、March 2009
8)　数学公式 III 特殊関数
森口繁一、宇田川銈久、一松信　2005 年　岩波書店
9)　High-precision calculation of multi-loop Feynman Integrals by difference equations
S. Laporta
J.Mod.Phys. A15:5087-5159(2000)
10) 高性能計算の扉
http://www.jicfus.jp/field5/jp/promotion/hpcdoor