スーパーコンピュータ有効活用の手引き
(平成23年度－平成27年度)

平成23年度から高性能計算の扉に記載しました各種計算機基本性能調査、各種計算機アプリケーション性能比較、多倍長計算手法のまとめとして性能関連事項を各種計算機性能調査まとめと、精度関連事項を多倍長計算手法まとめの2冊にしました。性能関連事項と精度関連事項は異なる視点からの検討が必要な部分が多くあるため、この2冊を列挙する体裁を取っています。論文リストはスライド238以降にまとめて掲載しています。

スーパーコンピュータ使用するケースは主に以下の
3つのケースに分類できます。

(1) 大学等での計算機教育や学問としての計算機論の展開
(2) 研究所等での論文作成の手段として
(3) 業務を行う公的機関や民間会社での手段として

一般的には (1) と (3) のサイトでは目的と手段が明確になっていますが (2) のサイトでは目的と手段が不明確で逆転している事が多々見られます

一例として過去ある大学で逆行列を求めるサブルーチン群を学内で公開した所,参照回数の50%以上がクラメルの公式で求めるサブルーチンだった事があります。
(2) や (3) の分野の人から見れば,時間もかかり,計算精度からみて問題のあるものを使用するのはおかしく見えますが、
教育という面からみると,連続変数を扱う数学と離散変数を扱う計算機の差や,計算機の演算量や精度を知る上では目的にかなっていると言えます。

今回は業務を行う場合から見る事にし、シュミレーションにおいて許容される精度で正しい結果を速く実行するためにスーパーコンピュータを使用するという観点からまとめています。
例えばスーパーコンコンピュータの性能と言う場合
性能（FLOPs）=演算量（FLOP）/実行時間（sec）
を重視しますが、ここでは実行時間（sec）で中心に見ます。
また実ジョブでは使用されない様な条件での性能は検討から外しています。すなわち中心におくのは、計算機学の研究ではなく,シュミレーションを実施する側にすると言う事です。

理由としては幾つか例をあげますが、

（1）フーリエ変換を使用するアプリケーションではFFTを使用すれば,実行時間は短縮されますが、性能の定義ではFFTのFLOPs値は低下します。
（2）最適化オプションを上げれば、一般的には実行時間は短縮されますがFLOPs値は低下します。
（3）スーパーコンピュータを使用するアプリケーションでは必ずしも浮動小数点演算だけとはかぎりません。

などです。

また実行時間に関しては、PCやサーバーから移植する場合には互換性などの問題も含め経過時間が問題となりますので測定結果はすべて経過時間としています。たとえ1CPUでの測定でもCPU時間ではなく経過時間としています。

各種計算機性能調査まとめ

平成23年度-平成27年度

目次

各種計算機性能調査まとめ

平成23年度-平成27年度

4.2 分子動力学計算
- 4.2.1 SR16000/M1 システム詳細
- 4.2.2 各種計算機性能比較

4.3 大規模疎行列の反復計算
- 4.3.1 対称問題
- 4.3.2 非対称問題

4.4 QCD計算

4.5 量子モンテカルロ法による物性スペクトル計算
- 4.5.1 SR16000/M1 システム
- 4.5.2 ieee754-2008形式

4.6 ファインマンループ積分計算
- 4.6.1 infra vtx計算
- 4.6.2 infra box計算
- 4.6.3 massless計算
- 4.6.4 4-6次元積分計算
  - 4.6.4.1 s221計算
  - 4.6.4.2 5次元積分計算
  - 4.6.4.3 6次元積分計算
- 4.6.5 3loop計算
- 4.6.6 4loop計算

1. はじめに

性能を考える場合,実行時間をもとにしても,
一般的に以下の事が言えます。

(1) 扱う問題がある程度特定される場合と
そうでない場合の性能は大きく異なる場合
がある。

(2) カタログ性能と実行時間は必ずしも
比例しない。

(3) 最適化オプションの効果には注意する必要
がある。

具体例を挙げると以下の様なものがあります。

(1)複素変数の絶対値計算

$$c = a + ib$$

$$|c| = \sqrt{a^2 + b^2}$$

a,bが絶対値がある範囲である事がわかっている場合は,実数演算で2つの乗算,1つの加算,1つの平方根計算で済む.$c = 10^{160} + i10^{160}$の場合,cは表現可能な数値で,結果も表現可能な値となるが、同じ計算をすると演算途中でオーバーフローが発生します。
これを防ぐため,関数ルーチンでは大小判定や除算等が使用されるのが一般的で数倍の性能差が出ます。

(2) カタログ性能は計算機に備わる演算器が
全て同時に動作する事を前提としているが,
実際のアプリケーションでは必ずしもそうは
ならない。倍精度演算と4倍精度演算の性能
比はSR11000では約10倍と言われてきたが
SR16000では4倍精度演算にはSIMD命令
が適用されないためその差は大きくなり、
15-20倍となっている。

(3) 最適化オプションでは結果が変わらない範囲
での最適化と、結果が変わる可能性がある
最適化のレベルがある。
特に,並列実行の際には,精度に敏感な
プログラムで総和,内積演算を含む場合には
性能以前に結果の検証が必要になります。

2.使用計算機

2.1 スーパーコンピュータ

(1) SR11000/K1(1ノード)
プロセッサ:power5
周波数:2.1GHz
CPUコア数 16
論理コア数 16
理論最大性能　134.4 GFLOPs
メモリ容量 32GB
メモリアーキテクチャー　Flat Memory　Interleave
L3キャッシュ　Off-Chip 36MB/2コア
演算器/物理コア　乗加算器2つ

(2)　SR16000/M1

プロセッサ:power7
周波数:3.83GHz
CPUコア数 32(物理的),64(論理的)
理論最大性能　980.48 GFLOPs
メモリ容量 256GB
メモリアーキテクチャー　NUMA,(16論理コア単位でflat)
SIMD(Single Instruction Multiple Data)を
サポートするVSX機構付き
L3キャッシュ　On-Chip 32MB/8コア
演算器/物理コア　乗加算器4つ

(3) SR16000/xm1

SR16000/xm1は周波数が3.3GHzで他はSR16000/M1と同じです。演算性能だけみれば,SR16000/M1 1ノードはSR16000/xm1の16%性能向上版ともいえます。またメモリ性能はSR16000/M1 の半分です。

この3機種では,(2)のSR16000/M1 がメインで(1)のSR11000/K1 は互換性,実行性能比を,SR16000/Xm1 はメモリ性能の影響と,smtの効果を見ています。

(4)BG/L
周波数　0.7GHz
1ノード　2　論理性能　5734.4 GFLOPs
L1キャシュ　32KB(データ,命令),L2キャシュ　2KB,L3キャシュ4MB

(5) BG/Q
周波数　1.6GHz
1ノード　16core　論理性能　204.8GFLOPs
L1 キャシュ　16/16KB (Core)
L2 32MB (node)
Main storage　16GB (Core)
Smt=1,2,4

(4),(5)に関してはメインは(5)で(4)は互換性を見ています。

(6) T2K 筑波システム

AMD quad-core Opteron 8000 シリーズ (Barcelona)

1 node:ピーク性能 147GFLOPs,16MPI/node

2.2 アクセラレータ

(1) HD5870

GPU カード型番:ATI RadeonHD5870
メモリ: GDDR5, 1 GB, 153.6 GB/s
ホストインタフェース: PCI Express 2.1 x16stream
processing unit: 3200個 (演算プロセッサ)
動作周波数: 850 MHz ピーク性能 (倍精度): 1088 Gflops

(2) HD6970

GPU カード型番:ATI RadeonHD6970
メモリ: GDDR5, 2 GB, 176 GB/s
processing unit: 6144個 (演算プロセッサ)
動作周波数: 880 MHzピーク性能 (倍精度):2703 Gflops

(3) グラフィックボード

HOST

E5－2670　2.60GHz　1cpu＝8core

キャシュ　20MB

2.6GHz×8×8＝166.4GFLOPs　2cpu

2cpu＝332.8GFLOPs

Xeon　Phi5110P　1.053GHz

60コア,4スレッド／1core

1.053GHz×60×4×4＝1010.88GFLOPs

2.3　サーバー

(1) x5570 8コア　2.93GHz

キャシュ　8MB/コア

(2) e5430 16コア　2.66GHz

キャシュ 12MB/コア

(3) 上記グラフィックボードのHOST

E5-2670もここに含めました。

3.基本演算

3.1 メモリバンド幅測定

メモリバンドテスト

copy	trans
do j = 1, N	do j = 1, N
do i = 1, N	do i = 1, N
a(i, j) = b(i, j)	a(i, j) = b(j, i)
end do	end do
end do	end do

性能測定結果(MB/sec)

N	演算	SR16000	xm1	E5-2670	Phi5110P
		32core	32core	16smp	240smp
8192	copy	216248	84717	25687	80130
	trans	3223	958	4692	8508
8193	copy	221165	78572	20112	78970
	trans	31387	7216	17959	10018

詳細　transは参照非連続

N=8192

	非連続	sr16000	sr16000	E5-2670	Phi5110P
	要素	32core	64smp	16smp	240smp
連続	なし	216248	189155	25687	80130
非連続	参照	3223	3082	4692	8508
非連続	格納	2282	3461	5103	6927
非連続	両方	871	1849	24912	10354

N=8193

	非連続	sr16000	sr16000	E5-2670	Phi5110P
	要素	32core	64smp	16smp	240smp
連続	なし	221165	182590	20112	78970
非連続	参照	31387	30856	17959	10018
非連続	格納	56743	52338	20496	16833
非連続	両方	54351	43623	18743	12710

xm1

N=8192

	非連続	smt2	smt2	smt4	smt4	smt4
	要素	32smp	64smp	32smp	64smp	128smp
連続	なし	84717	86216	84754	84639	71859
非連続	参照	958	961	960	948	937

N=8193

	非連続	smt2	smt2	smt4	smt4	smt4
	要素	32smp	64smp	32smp	64smp	128smp
連続	なし	78572	68711	78484	70198	79770
非連続	参照	7216	6328	6297	6078	6173

演算を含めた場合の結果

(MMAX=8193)

2D Copy

```
do j = 1,MMAX
do i = 1,MMAX
r(i,j)=p(i,j)
enddo
enddo
```

2D Scale

```
do j = 1,MMAX
do i = 1,MMAX
p(i,j)=scalar*q(i,j)
enddo
enddo
```

2D Add

```
do   j = 1,MMAX
  do i = 1,MMAX
  q(i,j)=p(i,j)+r(i,j)
  enddo
  enddo
```

2D Triad

```
do j=1,MMAX
do i=1,MMAX
r(i,j) = p(i,j) + scalar*q(i,j)
enddo
enddo
```

Transpose

```
do j=1,MMAX
do i=1,MMAX
r(i,j)=p(j,i)
enddo
enddo
```

SR16000/XM1とSR16000/M1の2機種で測定していますが,
約3倍の差があり,演算性能の差16%とは大きな差があります。
特に,Transpose　64スレッドでの差が顕著です。

メモリバンド幅性能測定結果(MB/sec)

SR16000/XM1 MB/sec

function	1スレッド	2スレッド	4スレッド	8スレッド	16スレッド	32スレッド	64スレッド
2D Copy	5070	8004	13786	27134	43057	68565	62532
2D Scale	4682	7982	13707	27011	41294	61917	58637
2D ADD	6563	11054	16604	32834	48544	78820	70376
2D Triad	7307	12097	16954	33525	49752	77751	70680
Transpose	426	725	1470	2649	4883	3801	2866

SR16000/M1 MB/sec

function	1スレッド	2スレッド	4スレッド	8スレッド	16スレッド	32スレッド	64スレッド
2D Copy	11661	23356	46292	92051	165536	138902	148588
2D Scale	12528	25012	49218	97630	170661	140657	163474
2D ADD	12918	25797	49839	97796	176648	151006	174514
2D Triad	12680	25216	48697	96034	174808	149709	171588
Transpose	1052	2055	4072	7971	15921	16231	30540

3.2 rump's　例題

rump's**例題とは,四則演算からなる簡単な計算で,有効ビット数が**121**ビット以上を要求する計算のため,精度、性能の検証によく使用されるものです。**

$$a = 77617.0, b = 33096.0$$

$$f = 333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b}$$

$$a^2 = 5.5b^2 + 1 = 6024398689$$

$$a^2(11a^2b^2 - b^6 - 121b^4 - 2) = -5.5b^2 - 333.75b^6 - 2$$

より, $f = -2 + \frac{a}{2b} = -\frac{54767}{66192} = -0.827396059946$

3.2.1 ieee754-2008系

拡張4倍精度：拡張倍精度変数を2つつなげたもの
6倍精度：倍精度変数を3つつなげたもの
8倍精度：倍精度変数を4つつなげたもの

rump例題テスト結果

e5430　実行結果一覧表

実行回数　100,000,000
演算量　1.9GFLOP　（各精度）

精度	言語	実行時間（秒）	smp実行時間 8smp(秒)	台数効果
拡張4倍精度	icc(strict)	45.5899	5.817464	7.84
拡張4倍精度	icc(extended)	37.5245	4.7933	7.83
6倍精度	icc	54.6475	6.9464	7.87
6倍精度	ifort	55.4023	6.9807	7.94
8倍精度	icc	125.8814	15.8532	7.94
8倍精度	ifort	126.4694	15.7271	8.04

t2k

実行回数60,000,000
演算量　1.14GFLOP　（各精度）

精度	言語	実行時間（秒）	smp実行時間 16smp(秒)	台数効果
拡張4倍精度	icc(strict)	58.88582	3.7665	15.63
拡張4倍精度	icc(extended)	50.962057	3.250428	15.68
6倍精度	icc	48.549858	3.132938	15.5
8倍精度	icc	113.161652	7.336832	15.42

拡張4倍精度はextended オプション効果あり。
（両CPUで）
Fortranとc++　での有意な性能差はない。
拡張4倍精度と6倍精度の性能は，
　e5430 は拡張4倍精度、T2kでは6倍精度が良いという逆の傾向がでています。

rump's　例題実行結果一覧

100,000,000回実行

実行時間(秒)

	C		FORTRAN	
精度	E5-2670	Phi5110P	E5-2670	Phi5110P
	16smp	240core	16smp	240core
拡張倍精度*2	1.949568	2.993835		
6倍精度	2.391048	3.373063	2.3545	3.3367
8倍精度	5.514834	5.949745	5.3994	5.8646
10倍精度			17.7158	15.0406

rump例題実行時間(秒)一覧表

整数演算方式

精度	E5-2670	Phi5110P
5倍精度	6.7659	20.3040
6倍精度	8.2706	24.5392
7倍精度	10.0500	28.9972
8倍精度	12.9452	35.6017
10倍精度	17.0668	52.6718
12倍精度	25.2205	78.2484
14倍精度	32.5928	102.1592
16倍精度	39.9565	127.1153

Phi5110PのThreads/core実行時間(秒)

改良版

threads	10倍精度	12倍精度
1	81.3013	124.3947
2	60.6723	92.7562
3	57.7494	86.7546
4	52.6718	78.2484

DD形式(一般サブルーチン使用)

精度	E5-2670	Phi5110P
	16smp	240core
10倍精度	15.8159	27.9302
12倍精度	27.7419	46.7320
14倍精度	46.8211	70.7831
16倍精度	72.6414	107.0264

E5－2670とPhi5110Pの性能に
関しては,DD形式では,8倍精度
演算で性能が逆転する。
またDD形式と整数演算方式では
E5－2670において10倍精度演算で
性能が逆転しています。
Phi5110Pでは,最適化オプションで
opt3とoptsで使用メモリ量が大きく変わる
ので,基本的にはopt3で実行するのが良い。
またレジスタの問題で整数演算方式では
演算量が多くなると,最適化が妨げられる
事があります。

E5-2660 rump's 例題実行時間一覧表

100,000,000回実行の実行時間
(実行時間:秒)

smp数	拡張倍精度*2	6倍精度	6倍精度	8倍精度	8倍精度
	c	c	F	c	F
1	27.615	38.205	33.162	96.057	78.723
2	13.852	19.391	16.626	48.815	39.568
4	7.248	10.081	8.607	25.221	20.414
8	4.315	5.367	4.643	13.546	10.986
16	3.038	3.729	3.517	7.131	8.288
32	2.257	3.562	2.755	7.803	6.412

拡張倍精度＊2 及び6倍精度FORTRANで32smpの場合の性能が良い。

3.2.2 POWER系

1スレッド実行

fma,nofma		SR16000/M1		
	1,000,000 回実行			
コンパイラ	f90	f90	xlf	xlf
精度	fma	nofma	fma	nofma
	(秒)	(秒)	(秒)	(秒)
4倍精度	0.241	0.249	0.15	0.21
6倍精度	0.514	0.453	0.31	0.39
8倍精度	0.937	0.849	0.56	0.73

8倍精度演算方式

QD方式:倍精度変数を4つつなげた方式
DQ方式:4倍精度変数を2つつなげた方式

(注)

(1) DQ方式はQD方式より精度が悪くなる。最悪の場合TD(倍精度変数を3つつなげたもの)と同じ精度になる。

(2) ieee754-2008でのDQ方式の有効ビット数は226ビットで,DD方式のQD方式より精度が良い。

Rump's例題8倍精度演算

100,000,000回実行の時間(秒)

SR16000/M1　1ノード実行

smp数	QD方式	DQ方式
1	99.22	334.01
2	52.95	168.78
4	31.49	86.83
8	36.55	47.55
16	45.68	42.06
32	54.52	47.67
64	49.09	43.3

4.2.3 超多倍長演算

多倍長rump's例題一覧表
1,000,000回実行

実行時間一覧表(秒)

E5-2670

演算精度	16smp	8smp	4smp	2smp	1smp
68倍精度	11.9698	23.8582	47.8585	95.5189	191.5384
128倍精度	42.3808	84.7253	169.0772	338.4619	677.3167
188倍精度	89.0890	177.8675	355.6048	711.6836	1425.8922
248倍精度	152.0543	303.7099	607.2627	1214.3922	2438.3971
308倍精度	248.2340				
368倍精度	355.4690				
428倍精度	476.9385				
488倍精度	617.0996				
548倍精度	833.8722				

Phi5110P　4smp/core

演算精度	240smp	180smp	120smp	60smp
68倍精度	46.3840	48.7955	50.8138	69.1542
128倍精度	156.0239	178.3118	181.1152	244.4755
188倍精度	331.2860	378.7431	384.5338	525.7136
248倍精度	585.7436	669.6312	694.6440	938.0996
308倍精度	966.5418			
368倍精度	1378.1645		68倍精度	
428倍精度	1859.8102		4smp	1smp
488倍精度	2410.4360		868.7135	3463.0626
548倍精度	3244.1159			

ともに並列化効果は出ていますが,E5-2670の効果が大きくなっています。

4.3　行列積,ストラッセン行列積

計算機の基本性能を見る演算としては行列積計算が適しています。
ここでは倍精度演算をSR16000/M1 SR16000/XM1,BG/Qでその特徴を検証し倍精度以外の行列積計算をサーバー系を含めてその性能を見ています。
ストラッセン行列積は行列積演算の演算量削減方法と知られていて,連立一次方程式の求解や後に示すルジャンドル陪関数を用いたいた計算にも適用が試みられています。
このためその演算量削減効果と実行時間短縮の効果をSR16000/M1を使用して検証しています。

3.3.1 行列積計算

3.3.1.1 SR16000/M1 システム

行列積計算を行う場合,行列のサイズNが小さい場合,3つの行列すべて合わせてもキャッシュに収まる範囲にあります。

行列積計算で$C = A^T B$を実行する場合の総メモリ所要量

N	実数型	複素数型
32	24KB	48KB
64	96KB	192KB
96	216KB	432KB
128	384KB	768KB
256	1536KB	3MB
512	6MB	12MB
768	13.5MB	27MB

赤色 L1D内に収まる。
緑色　L2内に収まる。
紫色 L3内に収まる。

N>100　（3桁）でキャッシュチューニングが重要となる。

キャシュチューニングの際,内積型計算を使用する場合
主要演算時の所要メモリ量N(N+2)＊8or16が
2MB以下に抑える事が重要になります。

主要演算時の所要メモリ量

実数型 $N \times (N+2) \times 8$ **バイト**

複素数型 $N \times (N+2) \times 16$ **バイト**

N	実数型	複素数型
31	8KB	16KB
63	32KB	64KB
64	33KB	66KB
100	80KB	160KB
192	291KB	582KB
256	516KB	1032KB
289	657KB	1314KB
384	1158KB	2316KB

$SR16000/M1$　1ノードで内積型行列積計算で
性能が出る条件

(1)$simd$命令が適用される。
(2)最内側ループ(k)内の演算で24の乗加算命令が適用される
(3)最外側ループ(j)のループ長が$4\times32=128$の倍数
(4)中間のループ(i)のループ長が6×2^nの倍数
(5)主要演算時の所要メモリ量が$2MB$以下

実数型ではN=384が上記すべての条件を満たす。
複素数型ではすべてをみたすサイズはないので
Nに応じて最適なものを探します。

これらの事からNが3桁でも実数型ではN＝504,
複素数型ではN＝360を境にチューニング手法を
変える必要が出てきます。

実数型行列積計算

実数型行列積計算性能一覧表

内積タイプとは,以下の3重*DO* ループで

DO $j = 1, n, lj$

DO $i = 1, n, li$

DO $k = 1, n$

***lj* * *li*で示しています.**

1ノード smt=off,smp=32 実行結果一覧

N	内積タイプ	GFLOPs
32	1*16	425.109
64	2*8	645.747
96	3*8	725.319
128	4*4	807.875
128	4*8	807.924
192	6*6	827.583
288	3*6	800.614
384	4*6	860.561

定義式どうりのソースでの結果(lj=1,li=1)

N	32smp	64smp
31	105.78	91.080
61	145.624	221.050
67	210.445	211.297
97	253.611	320.202
127	333.003	424.066

性能向上要件(5)のテスト

$N \times (N + 2) \times 8$バイト $\leq 2MB$
での実数型行列積計算
の性能

1ノードsmt=off,32smp

内積タイプ	N=480	N=504
4*6	819	833.002
6*6	739	780.324
4*8	813	849.898
4*4	827	831.878

複素数変数行列積計算

複素数型行列積計算性能一覧表

内積タイプとは,以下の3重*DO* ループで

DO $j=1,n,lj$

DO $i=1,n,li$

DO $k=1,n$

$lj*li$**で示しています**.

1ノード smt=off,smp=32 実行結果一覧

N	内積形式	GFLOPs
32	1*16	227.796
64	2*4	454.764
64	2*8	451.572
96	3*4	602.258
128	4*4	470.995
192	3*6	677.963
192	3*4	719.704
192	6*3	817.149
288	9*1	774.796
288	3*6	739.758
288	3*4	809.752

定義式どうりのソースでの結果(lj=1,li=1)

N	32smp	64smp
31	86.505	76.377
61	254.475	305.272
67	191.442	182.165
97	206.989	262.580
127	273.559	348.667

3.3.1.2 xm1システム

smt=4の効果検証

検証は複素数変数行列積と実数変数行列積計算で
$simd, nosimd$で行いました。

(1)複素行列積$C = (A^T B)^T \ \bar{A}$を28000回計算する。

定義式どうりに計算する複素数変数プログラム$CMULT$

実数変数でて定義どうり計算するプログラム$RMULT$

$RMULT$は$simd, nosimd$で計算

(2)実行列積$C = A^T B$を計算する。

実行回数は$N = 288,384$　　100,000回

$N = 480,504$　　10,000回

アンローリング数は最外側4,中間6

プログラムは$MULTIN$

実数変数行列積計算$C = A^T B$

multin　4*6タイプ　simd

N=384,演算量=11325GFLOP
実行時間(秒)一覧表

smp数	実行時間	GFLOPs
32	16.436	689
64	34.195	331
96	24.207	468
128	25.639	442

N=288　演算量＝4478GFLOP
実行時間(秒)一覧表

smp数	実行時間	GFLOPs
32	10.116	443
64	18.239	246
72	14.379	311
96	13.047	343
128	17.722	253

N=480　演算量＝2212GFLOP
N=504　演算量＝2560GFLOP
実行時間(秒)一覧表

smp	n=480	n=480	n=504	n=504
	実行時間	GFLOPs	実行時間	GFLOPs
32	3.477	636	3.802	673
64	5.626	393	5.816	440
96	6.898	321	7.566	338
128	6.055	365	6.650	385

最高性能一覧表

N	GFLOPs	実行効率(%)
288	443	52
384	689	82
480	636	75
504	673	80

最高性能はSR16000/M1と同じN=384で達成

32smpでのsmt=2,smt=4の性能比較

$smt=2$と$smt=4$での$32smp$性能比較

内積タイプ 4*6

N	smt=2			smt=4		
	演算量	実行時間	性能	演算量	実行時間	性能
	(GFLOP)	(秒)	(GFLOPs)	(GFLOP)	(秒)	(GFLOPs)
288	4478	11.301	396	4478	10.116	443
384	11325	17.645	643	11325	16.436	689
408	1358	2.651	512	1358	2.771	490
432	1612	2.838	568	1612	2.927	488
456	1896	3.626	523	1896	3.394	559
480	2212	3.638	608	2212	3.599	615
504	2560	4.02	637	2560	3.802	673
528	2944	5.39	546	2944	5.351	560
768	9060	66.18	137	9060	58.292	155

内積タイプ 3*4

N	smt=2			smt=4		
	演算量	実行時間	性能	演算量	実行時間	性能
	(GFLOP)	(秒)	(GFLOPs)	(GFLOP)	(秒)	(GFLOPs)
192	1416	3.269	433	1416	3.008	471
288	4478	12.048	372	4478	8.869	505

複素変数行列積計算 $C = (A^T B)^T \bar{A}$

N=289,演算量=10814GFLOP
実行時間(秒)一覧表

smp数	rmult	rmult	cmult	cmult
simd	on	off	on	off
32	44.887	193.428	71.381	50.786
64	39.372	206.027	58.254	60.995
97	46.455	84.652	62.35	51.837
128	64.086	84.476	72.36	52.887

N=288,演算量=10702GFLOP
実行時間(秒)一覧表

smp数	rmult	rmult	cmult	cmult
simd	on	off	on	off
32	31.299	87.594	56.235	87.081
64	36.029	107.324	58.964	67.362
96	30.183	159.467	46.902	48.036
128	34.372	110.544	55.66	86.615

N=31,演算量=13.346368GFLOP
実行時間(秒)一覧表

smp数	rmult	rmult	cmult	cmult
	on	off	on	off
32	0.124	0.176	0.189	0.193
64	0.155	0.205	0.212	0.212
96	0.195	0.236	0.395	0.219
128	0.321	0.219	0.434	0.261

Smp数32,96が良い値となっています。

3.3.1.3 BG/Qシステム

複素数変数行列積計算

$C^T = A^T B$

の計算を複素数変数のままと、実数変数に直した場合を実行。

SIMD命令を適用させるため,

－qstrict　オプションは外しています。

測定条件は

1ノード　openmpによる並列化

bg　複素変数行列積

$C^T = A^T B$

-O5 -qarch=qp -qtune=qp -qcache=auto
-qhot=level=2 -qipa=level=2 -qsimd=auto
-qsmp=omp

複素数型

実行時間一覧表(秒)

n	演算量 (GFLOP)	タイプ	16smp	32smp	48smp	64smp
32	734	1*16	78.359	97.998		
64	587	2*8	25.179	22.174		
96	198	3*4	4.094	3.930	3.831	
128	470	2*4	10.283	11.250		8.200
192	1585	3*4	32.378	25.078	22.773	22.917
288	5351	3*4	102.125	64.950	70.568	
384	12684	3*4	231.934	154.786		156.162

(注)48smpでは,96,192はタイプ2*4にしている。

最高性能は82GFLOPs

bg　　複素変数行列積

$C^T = A^T B$

-O3 -qarch=qp -qtune=qp -qcache=auto -qhot=level=1
-qipa=level=1
-qsimd=auto -qsmp=omp

複素数型

実行時間一覧表(秒)

n	演算量 (GFLOP)	タイプ	16smp	32smp	48smp	64smp
96	198	3*4	4.075	3.448	3.673	
128	470	2*4	10.155	7.928		8.123
192	1585	3*4	32.373	20.034	22.314	20.523
256	3758	2*4	81.983	49.812		57.069
288	5351	3*4	100.029	63.432	70.078	
384	12684	3*4	238.699	153.374		155.539

(注)48smpでは,96,192はタイプ2*4にしている。

性能は84GFLOPs,83GFLOPsと-O5の場合より少し良くなっています。(誤差の範囲)

bg　　複素変数行列積

$C^T = A^T B$

-O5 -qarch=qp -qtune=qp -qcache=auto
-qhot=level=2 -qipa=level=2 -qsimd=auto
-qsmp=omp

実数型

実行時間一覧表(秒)

n	演算量 (GFLOP)	タイプ	16smp	32smp	48smp	64smp
32	734	1*16	125.247	124.539		
64	587	2*8	19.187	20.635		
96	198	3*4	4.894	4.909	5.200	
128	470	2*4	10.513	10.522		10.205
192	1585	3*4	43.199	33.340		24.860
288	5351	3*4	123.406	106.704	87.407	
384	12684	3*4	263.06	256.902		187.866

(注)48smpでは,96,192はタイプ2*4にしている。

ソースはSR16000/M1,XM1と同じものを使用。
性能は64GFLOPs,61GFLOPs,68GFLOPsと
複素変数の場合に比べて有意に劣っています。

3.3.1.4 各種演算精度の 行列積計算

比較はSR16000 1ノード(32core,64smp),ES-2670 16smp,
Phi5110P 240core　で実施しています。最も多くのケースを
実施したのは行列サイズN=240,N=480の1000回実行です。
Cは,ES-2670,Phi5110Pで倍精度,拡張倍精度,拡張倍精度
＋拡張倍精度のケースを行っています。名称はそれぞれ
Double,exdouble,ddexdoubleとしています。FORTRANは
SR16000,ES-2670,Phi5110Pで倍精度multd,4倍精度(ieee754
-2008データ形式) multq,dd形式4倍精度ddmultd,dd形式の
6倍精度,8倍精度,10倍精度,それぞれmult6,mult8,mult10
です.SR16000では特にmultd,multqは実施していません。
また比較という事で,テストプログラムは定義式どうりの
コーデイングでアンローリング,キャシュチューニング等は
実施していません。演算量は以下の様になっています。

演算量(GFLOP)一覧

プログラム	N=240	N=480
multd	28	221
ddmultd	470	3760
mult6	1849	14789
mult8	3866	30928
mult10	8657	69265

行列サイズN=240の結果は以下の様になっています。

	行列積N=240,1000回実行の性能比較			
		実行時間(秒)		
プログラム	SR16000		phi5110P	E5-2670
	32core	64smp	240smp	16smp
multd			1.824	0.403
ddmultd	7.360	3.753	2.952	5.618
multq			54.771	26.297
double			1.941	0.421
exdouble			5.735	2.663
ddexdouble			71.820	55.966
mult6	21.274	14.789	20.972	26.144
mult8	37.599	24.575	33.496	56.351
mult10	79.776	53.658	86.748	210.930

(1) dd形式4倍精度はphi5110Pは高速
(2) dd形式6倍精度,8倍精度,10倍精度はSR16000 32coreとphi5110Pの性能が同等。
(3) Cとソフトウエアサポートルーチンは E5-2670が高速。

行列サイズN=480の結果は以下の様になっています。

	行列積N=480,1000回実行の性能比較			
		実行時間(秒)		
プログラム	SR16000		phi5110P	E5-2670
	32core	64smp	240smp	16smp
multd			3.445	3.204
ddmultd	55.214	29.658	11.212	44.669
multq			469.772	209.251
double			3.728	3.196
exdouble			31.246	21.076
ddexdouble			256.872	191.585
mult6	164.514	119.144	159.443	205.080
mult8	289.634	196.611	285.541	460.643
mult10	605.880	436.391	688.837	1686.164

性能比較の傾向はN=240の場合と全く同じ。

各種行列積計算の性能は以下の様になっています。

各種行列積計算性能比較表

実行時間一覧表(秒)

N	CPU	double	exdouble	ddexdouble	multd	ddmultd
240	E5-2670	0.421	2.663	55.966	0.403	5.618
	Phi5110P	1.941	5.735	71.820	1.824	2.952
480	E5-2670	3.196	21.076	191.585	3.204	44.669
	Phi5110P	3.728	31.246	256.872	3.445	11.212
720	E5-2670	17.364	70.782	485.278	19.230	150.859
	Phi5110P	11.141	103.516	948.543	10.315	35.615
960	E5-2670	40.647	204.819	1035.046	45.322	357.364
	Phi5110P	25.932	253.483	2284.650	25.001	116.462
1200	E5-2670	79.576	369.371	1890.814	88.222	720.564
	Phi5110P	56.584	564.900	4642.316	57.053	241.207

(1) 倍精度ではc,FORTRANともN=720でPhi5110P　の性能がE5-2670を上回ります。
(2) 拡張倍精度はすべてE5-2670の性能が　　上回っています。
(3) SR16000の4倍精度では,Phi5110Pが非常に効力を発揮する事がわかります。

これに関連した計算は以下の様になっています。

4倍精度行列積計算

N=256　1回の実行時間

SR16000 singleジョブ
　92.638 msec　　ieeequad 8704.767 msec

x5570
　double+double　　457.259 msec
　extend double +extend double 734.146 msec

e5430 gcc
　double+double　　652.392 msec
　extend double +extend double 759.277 msec

e5430 icc
　double+double　　309.527 msec
　extend double +extend double 621.636 msec

Phi5110P　240smpがSR16000 1ノードより高速という結果と合わせPhi5110Pが有効な事がわかります。

6倍精度,8倍精度,10倍精度行列積計算

SR16000/M1　　1ノード実行
コンパイルオプションは
-WO,' opt (o (s) ,disbracket (0)) '
opt (o (ss)) にしたり、disbracket (0) を外すと結果不正となります。

行列積テスト結果一覧表

N=100,1000回実行時間(秒)

精度	single	32core	64smp
6倍精度	46.315	1.877	1.304
8倍精度	80.729	3.285	2.14
10倍精度	188.761	7.58	5.275

(1) 並列化効果はよくでている。
(2) 8倍精度はSIMDがよく効いているので他の2つの精度に比べ演算数比率より性能が良い。
(8倍精度は4つの倍精度変数を扱うため。)

4.3.2 ストラッセンの行列積

代表的な例でその計算方法と演算量を示しました。

$$C = AB$$

$$A = \left(\frac{A_{11}}{A_{21}} \middle| \frac{A_{12}}{A_{22}} \right), B = \left(\frac{B_{11}}{B_{21}} \middle| \frac{B_{12}}{B_{22}} \right), C = \left(\frac{C_{11}}{C_{21}} \middle| \frac{C_{12}}{C_{22}} \right)$$

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_2 = (A_{21} + A_{22})B_{11}$$

$$P_3 = A_{11}(B_{12} - B_{22})$$

$$P_4 = A_{22}(B_{21} - B_{11})$$

$$P_5 = (A_{11} + A_{12})B_{22}$$

$$P_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$C_{11} = P_1 + P_4 - P_5 + P_7$$

$$C_{12} = P_3 + P_5$$

$$C_{21} = P_2 + P_4$$

$$C_{22} = P_1 + P_3 - P_2 + P_6$$

m段目の行列積の回数M_m,**行列和の回数**A_m

$$M_0 = 1, A_0 = 0$$

$$M_m = 7M_{m-1}, A_m = 18M_{m-1} + A_{m-1}$$

$$M_m = 7^m, A_m = 3(7^m - 1)$$

もとの行列のサイズN,**m段目は**$\frac{N}{2^m}$

演算量　通常の行列積$2N^3$

strassen　$2(\frac{7}{8})^m N^3 + \frac{3(7^m - 1)}{4^m} N^2$

演算量比 $= \frac{N}{(\frac{7}{8})^m N^3 + \frac{3}{2}\frac{(7^m - 1)}{4^m} N^2} \fallingdotseq \frac{8^m N}{7^m (N + 3 \times 2^{m-1})} \fallingdotseq (\frac{8}{7})^m$

strassenの**行列積の演算量は通常の行列積の**$(\frac{7}{8})^m$

となり,**数学的には**,strassenの**行列積の演算量は**

$N^{2.807}$**に比例すると言えます。**

演算量に関して

strassenの行列積　性能一覧表

SR16000/M1

一Oss　N＝65536　smt on 64smp/32core

プログラムの性能(GFLOPs)

演算量$2\times N^3$(FLOP)/**実行時間(秒)**

段数	プログラム（GFLOPs）	モニター（GFLOPs）	実行時間（秒）
0	304.764	305.839	1847.162
1	343.957	302.132	1636.684
2	377.379	290.097	1491.735
3	407.541	274.253	1381.331
4	454.6	267.381	1238.34
5	556.783	287.836	1011.074
6	461.391	210.794	1220.112
7	300.128	121.916	1875.696

見かけ上の性能値と論理最大性能

ストラッセン行列積性能一覧表

SR16000／M1　　smt=off

測定値(GFLOPs)

演算量$2\times N^3$(FLOP)/**実行時間(秒)**

N	段数	smp	測定値 (GFLOPs)	論理最大性能 (GFLOPs)
38400	4	8	254.12	245.12
38400	4	32	536.70	980.48
46080	3	8	255.76	245.12
46080	3	32	726.88	980.48
61440	4	8	307.47	245.12
61440	4	12	408.76	367.68
61440	4	16	589.95	490.24
61440	4	20	586.77	612.80
61440	4	24	600.06	735.36
61440	4	28	578.24	857.92
61440	4	32	1027.31	980.48
61440	5	8	292.72	245.12
61440	5	16	510.21	490.24

4.実アプリケーション

4.1 重力多体問題（N体問題）

重力多体問題は

$$\vec{F}_i = Gm_i \sum_{j \neq i} \frac{m_j}{r_{ij}^3} \vec{r}_{ij} \qquad r_{ij} = |\vec{r}_i - \vec{r}_j|$$

G：万有引力定数, m_i：粒子iの質量

$\vec{r}_i$：粒子iの位置, $\vec{F}_i$：粒子iにかかる力

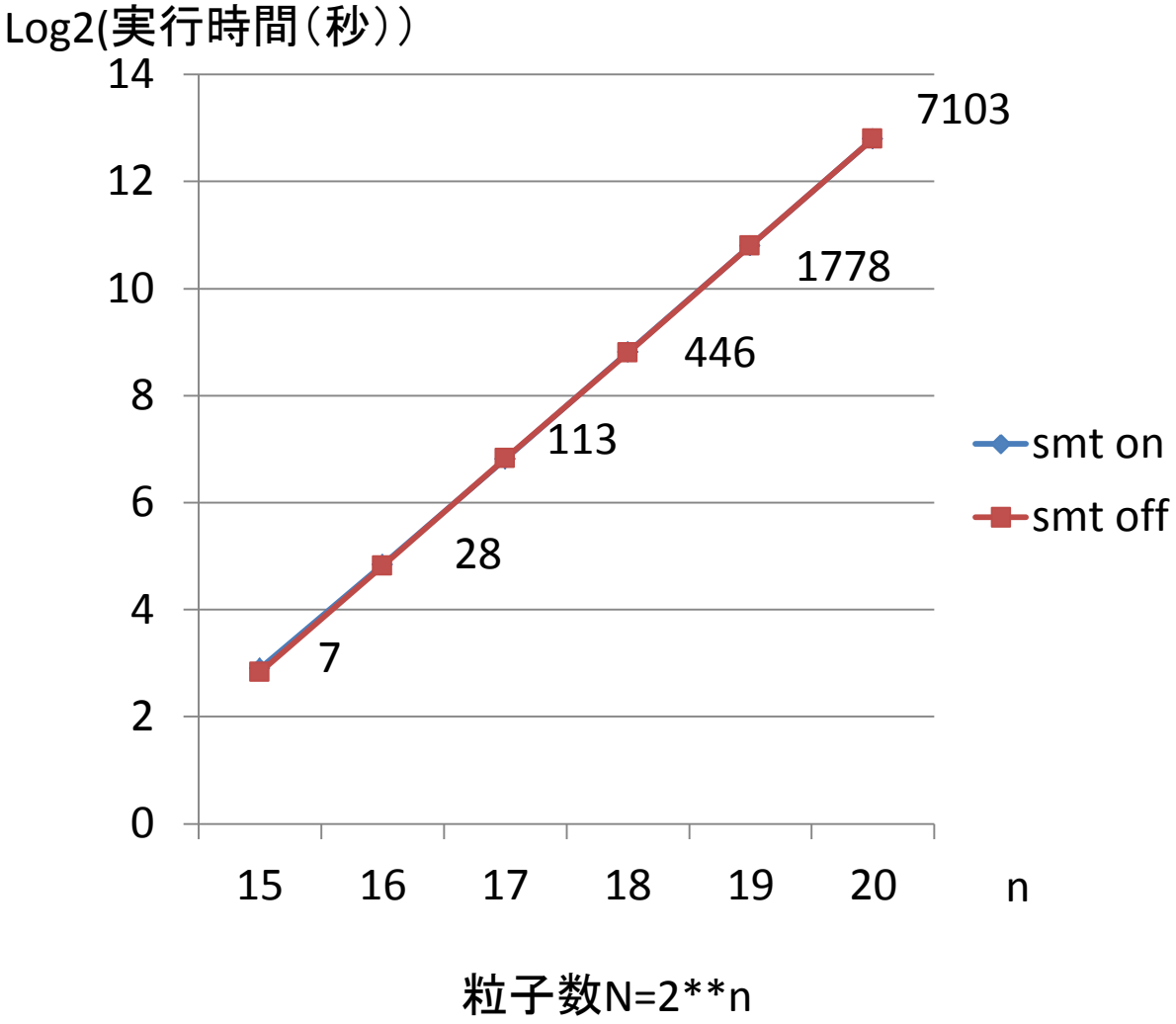
で計算します。

演算精度は倍精度で*SIMD*命令が適用されます。

演算量は粒子数Nの二乗に比例します。

4.1.1 SR16000/M1 システム

4ノードでフラットMPIでSMT　ON、SMT　OFFの場合の結果です。演算量がN=2**nの二乗に比例しますので、実行時間はlog2スケールで表しています。タイムステップ数は100です。



SMT ON,OFF での実行時間に有意差は見られません。

また、フラットMPIとハイブリッドの差をN=2**18,2**20で調べた結果は以下の様になりました。

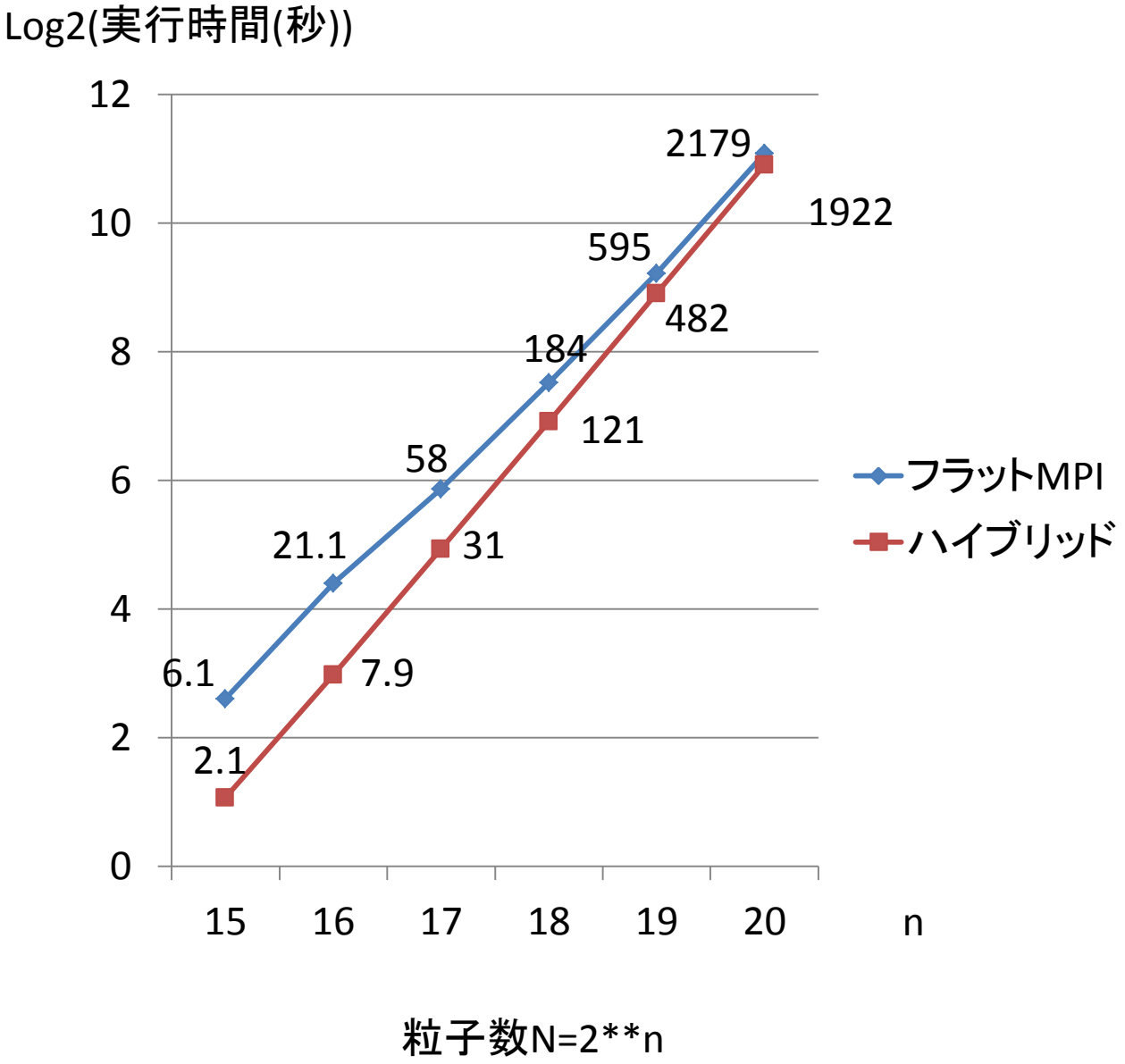
サイズ2**n	smp数	task数	ノード数	実行時間(秒)
20	1	256	8	3599
20	1	512	8	3612
20	32	8	8	3603
20	64	8	8	3604
20	8	32	8	3593
20	8	64	8	3607

サイズ2**n	smp数	task数	ノード数	実行時間(秒)
18	1	256	8	224
18	1	512	8	231
18	32	8	8	222
18	64	8	8	228
18	8	32	8	222
18	8	64	8	223

フラットMPIとハイブリッドでも実行時間に有意差は見られません。

4.1.2 BG／Q　システム

領域分割をせずに並列実行する場合、粒子数N=2**15=32768からN=2**20=1048576 とすると1スレッド当たりの所要メモリ量が3.328MBから104MBため、SMPとMPIのハイブリッド方式がフラットMPIより効果がでます。演算量がNの二乗に比例するため、実行時間の軸はlog2を取っています。実行結果は以下の様になりました。タイムステップ数は100です。



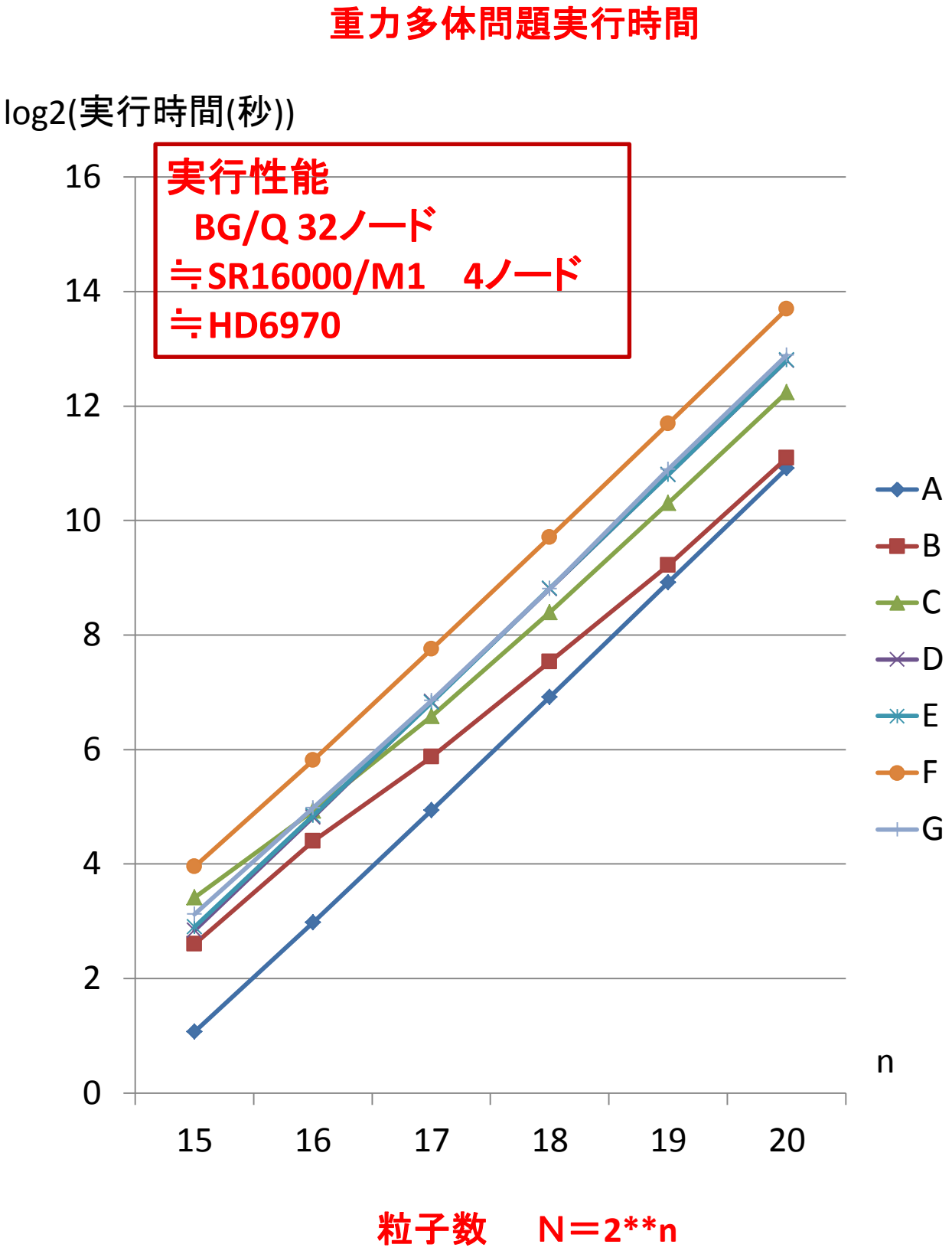
4.1.3 各種計算機性能比較

使用した計算機は以下のものです。

	計算機						
	A	BG/Q 128node smp					
	B	BG/Q 128node 8192MPI					
	C	BG/Q 32node 2048MPI					
	D	SR16000/M1 4node 128MPI					
	E	SR16000/M1 4node 256MPI					
	F	HD5870	3200core				
	G	HD6970	6144core				
	HD6970	880MHz	1536core	683GFLOPs	176GB/sec		
	HD5870	850MHz	1600core	544GFLOPs	153.6GB/sec		

扱ったケースは,
演算量は可変で粒子数を変化させた場合と
演算量は一定で粒子数を変化させた場合を
扱っています。

4.1.3.1 演算量可変ケース



5.1.3.2 演算量一定ケース

粒子数N＝1000,4000,10000で
タイムステップ数はそれぞれ,10000,
625,100として演算量を同じにして
(平方根計算あり270GFLOP,なし580GFLOP)
性能を比較.演算は倍精度演算。
N＝1000の場合,並列化オーバーヘッドにより
SR16000の性能が勝っているが,
N＝4000,10000ではGPUの性能が勝っています。

またieee754-2008系の特徴でもある
拡張倍精度演算に関する結果も扱っています。

N体問題実行結果一覧表(秒)

粒子数=1000,タイムステップ=10000
演算量(ソース)=500GFLOP

SR16000　1ノード

32core	64smp	最適化
4.7332	4.7668	OPT3
2.6379	2.6471	SOPT

E5-2670	16smp	16.0409

各種加速装置実行時間一覧表

GPU	1ボード (1smp/core)	2ボード (2smp/core)	3ボード (3smp/core)	4ボード (4smp/core)
HD5870	24.0556	32.9047		
HD6970	30.4342	42.0839	52.2729	64.2911
HD7970	41.0905			
W8000	41.0614			
HD7980	47.1802			
Phi5110P	4.2137	3.1891	2.8292	2.6806

N体問題実行結果一覧表(秒)

粒子数=4000,タイムステップ=625
演算量(ソース)=500GFLOP

SR16000　1ノード

32core	64smp	最適化
4.2597	4.3365	OPT3
2.6304	2.7826	SOPT

E5-2670	16smp	4.6922

各種加速装置実行時間一覧表

GPU	1ボード (1smp/core)	2ボード (2smp/core)	3ボード (3smp/core)	4ボード (4smp/core)
HD5870	4.6126	5.2407		
HD6970	5.2541	6.0501	6.8695	7.7215
HD7970	7.8925			
W8000	8.0438			
HD7980	8.7800			
Phi5110P	3.1736	1.9231	1.6821	1.3974

N体問題実行結果一覧表(秒)

粒子数N=10000,タイムステップ=100
演算量(ソース)=500GFLOP

SR16000　1ノード

32core	64smp	最適化
4.1637	4.5797	OPT3
2.5880	2.6867	SOPT

E5-2670	16smp	4.5523

各種加速装置実行時間一覧表

GPU	1ボード (1smp/core)	2ボード (2smp/core)	3ボード (3smp/core)	4ボード (4smp/core)
HD5870	2.7480	1.9917		
HD6970	3.0483	2.1885	2.3725	2.5931
HD7970	3.0059			
W8000	3.0796			
HD7980	3.2465			
Phi5110P	3.1736	1.8653	1.5618	1.3542

拡張倍精度演算

N体問題 拡張倍精度実行時間一覧表(秒)					
N	反復回数	E5-2670	Phi5110P	E5-2660	E5-2660
		16smp	240smp	16smp	32smp
1000	10000	15.8442	32.0745	20.6893	22.7324
4000	625	10.1019	28.8471	18.4697	14.5187
10000	100	10.0968	28.5530	11.1613	15.1299

倍精度の場合とは逆にE5-2670の性能がE5-2660の性能を上回っています。

拡張倍精度に対する処理でE5-2670のコンパイラがe5430のコンパイラより非常に適している事によります。

4.2 分子動力学計算

分子動力学計算は以下の2つの計算からなり、
クーロン力計算は重力多体問題と同様の計算と
なりますが、異なる計算として、
ワンデルワールス力計算があります。逆二乗則と
逆六乗則の差により、演算する範囲が異なり、
演算量も単純に粒子数の何乗に比例するとは
言えません。
演算精度は倍精度でSIMD命令は一部適用されます。

クーロン力 $\vec{F}_i = \frac{q_i}{4\pi\varepsilon_0} \sum_{j \neq i} \frac{q_j}{r_{ij}^3} \vec{r}_{ij} \qquad r_{ij} = |\vec{r}_i - \vec{r}_j|$

q_i : 電荷量, ε_0 : 真空の誘電率

ワンデルワールス力

分子間に働く分散力で、等方向性で原子間
距離の6乗に反比例する力

$$F = k \times \frac{\alpha_a \alpha_b}{r^6} \qquad \alpha : \text{分極率}$$

4.2.1 SR16000/M1 システム詳細

SR16000/M1　複数ノードでの実行時間は以下の様になっています。実行はフラットMPIを使用しています。所要メモリはそれぞれ、11.8MB,28.0MB,54.7MBとなっています。

実行時間（単位秒）

N=48

MPI数	ノード数	VdW	Coulomb	全体
128	4	1.8198	4.44435	6.26732
256	4	2.52998	8.61865	11.15244
256	8	1.71889	2.66658	4.38859
512	8	3.35877	3.98576	7.34833

N=64

MPI数	ノード数	VdW	Coulomb	全体
128	4	12.54277	63.21624	75.76465
256	4	16.55753	97.01006	113.57417
256	8	8.77493	35.47022	44.25077
512	8	20.90441	63.69971	84.61077

N=80

MPI数	ノード数	VdW	Coulomb	全体
128	4	44.93113	412.63567	457.57689
256	4	63.25216	362.52039	425.78437
256	8	33.06468	259.85763	292.93251
512	8	91.30541	281.84471	373.16153

粒子の数はn=N**3 で演算量はCoulmb(クーロン力)はn(n-1)/2に比例,VdW(ファンデルワールス力)はn(n-1)/2/近傍にある粒子の数に比例します。

VdWではすべてのケースでSMT OFF の効果があり、クーロン力はN=80,4node ではSMT　ONがそれ以外はSMT　OFFが効果があります。これは転送量がMPI数に比例するため、演算量との比でSMT　ON,OFFの効果が異なる様になります。

4.2.2 各種計算機性能比較

N	所要メモリ（MB）
48	11.8125
64	28
80	54.6875
96	94.5
112	150.625
128	224

実行時間（秒）一覧表

N=48

SR16000／M1

MPI数	ノード数	VdW	Coulomb	全体
256	4	2.52998	8.61865	11.15244
256	8	1.71889	2.66658	4.38859
512	8	3.35877	3.98576	7.34833

BG／Q

MPI数	ノード数	VdW	Coulomb	全体
512	32	8.16086	8.81916	16.98838
1024	32	11.82922	9.99511	21.83329
2048	32	24.60092	17.70815	42.32042

N=64

SR16000／M1

MPI数	ノード数	VdW	Coulomb	全体
128	4	12.54277	63.21624	75.76465
256	4	16.55753	97.01006	113.57417
256	8	8.77493	35.47022	44.25077
512	8	20.90441	63.69971	84.61077

BG／Q

MPI数	ノード数	VdW	Coulomb	全体
512	32	29.56434	37.76663	67.34834
1024	32	32.48871	32.1485	64.65611
2048	32	57.664	45.01047	102.69909

N=80

SR16000／M1

MPI数	ノード数	VdW	Coulomb	全体
128	4	44.93113	412.63567	457.57689
256	4	63.25216	362.52039	425.78437
256	8	33.06468	259.85763	292.93251
512	8	91.30541	281.84471	373.16153

BG／Q

MPI数	ノード数	VdW	Coulomb	全体
512	32	92.13976	128.87558	221.04942
1024	32	80.97082	91.95302	172.96071
2048	32	121.47084	106.98893	228.50708

メモリの制限により,N=112,N=128ではBG/Q 2048MPIは実行不可

N=96

SR16000/M1

MPI数	ノード数	VdW	Coulomb	全体
128	4	135.30229	1018.25815	1153.57737
256	4	170.37763	1035.44061	1205.83632
256	8	96.82403	592.82008	689.65997
512	8	217.66733	688.59726	906.2826

BG/Q

MPI数	ノード数	VdW	Coulomb	全体
512	32	247.50348	392.02168	639.58082
1024	32	185.35274	274.96719	460.38124
2048	32	231.82941	340.60404	572.51359

N=112

SR16000/M1

MPI数	ノード数	VdW	Coulomb	全体
128	4	333.35061	2736.97492	3070.34942
256	4	411.22115	2556.15124	2967.40039
256	8	244.40453	1447.09706	1691.52595
512	8	369.37199	1905.33538	2274.73516

BG/Q

MPI数	ノード数	VdW	Coulomb	全体
512	32	591.7365	1262.05753	1853.88598
1024	32	394.3729	1181.64034	1576.11295

N=128

SR16000/M1

MPI数	ノード数	VdW	Coulomb	全体
128	4	755.4252	5581.71674	6337.17718
256	4	924.26017	6382.94117	7307.24141
256	8	539.1269	3196.45186	3735.61507
512	8	750.14716	4215.49161	4965.67888

BG/Q

MPI数	ノード数	VdW	Coulomb	全体
512	32	1317.87477	3370.73599	4688.74093
1024	32	805.80939	3364.92311	4170.8758

BG/Q

mdcore

smp数	ノード数	VdW	Coulomb	全体	対smt=1
16	32	219.958	326.512	546.518	1
32	32	133.103	198.099	331.25	1.65
48	32	113.692	166.067	279.81	1.95
64	32	110.894	158.311	269.255	2.03

mdcore2

smp数	ノード数	VdW	Coulomb	全体	対smt=1
16	32	232.7985	361.71276	594.61322	1
32	32	119.5483	186.49663	306.14688	1.94
48	32	80.8	140.882	221.784	2.68
64	32	61.61527	109.81348	171.53069	3.47

xm1

mdcore

smp数	ノード数	VdW	Coulomb	全体	対smt=1
32	1	370.355	1031.531	1402.025	1
64	1	246.078	611.013	857.267	1.64
96	1	223.676	446.46	670.328	2.09
128	1	173.189	340.836	514.222	2.73

mdcore2

smp数	ノード数	VdW	Coulomb	全体	対smt=1
32	1	353.256	930.879	1254.276	1
64	1	306.571	605.227	911.972	1.38
96	1	229.255	442.445	671.89	1.87
128	1	177.715	339.45	517.357	2.42

MPIオーバーヘッド削減の効果はBG/Qででています。
Xm1では1ノード並列のためほとんど変化はありません。

smt=1に対する効果一覧表

プログラム	cpu	smt=2	smt=3	smt=4
mdcore	BG/Q	1.65	1.95	2.03
mdcore2	BG/Q	1.94	2.68	3.47
mdcore	xm1	1.64	2.09	2.73
mocore2	xm1	1.38	1.87	2.42

xm1
論理コア128(new),64(old)の比較

mdcore

smp数	ノード数	VdW	Coulomb	全体
32 new	1	370.355	1031.531	1402.025
32 old	1	370.239	1051.356	1421.733
64 new	1	246.078	611.013	857.267
64 old	1	250.312	554.729	805.976
128 new	1	173.189	340.836	514.222

mdcore2

smp数	ノード数	VdW	Coulomb	全体
32 new	1	353.256	930.879	1254.276
32 old	1	352.106	931.449	1283.693
64 new	1	306.571	605.227	911.972
64 old	1	297.089	556.781	854.376
128 new	1	177.715	339.45	517.357

BG/Q　MPIオーバーヘッド削減の効果が大きい（smt=4）
Xm1　論理コア128（smt=4）の拡大の効果が大きい。

4.3 大規模疎行列の反復計算

4.3.1 対称問題

倍精度演算

ポアソン方程式

$\Delta u = -f \quad (\Omega), u = 0 \quad (\partial\Omega)$

$\Omega = [0,1] \times [0,1] \times [0,1]$

$N = 200 \times 200 \times 200$

収束判定値: 共役残差 10^{-12}

$u = \sin(\pi x)\sin(\pi y)\sin(\pi z)$

$f = 3 \times \pi^2 \times \sin(\pi x)\sin(\pi y)\sin(\pi z)$

(注)

初期値は $x(i) = 1.0$

$cgs, cgsilu, cgsmil$**は収束しないので** $x(i) : (0,1)$**の一様乱数値.**

$cgsm$**は** $cgsilu$**を,** $cgsmm$**は** $cgsmil$**を並列化して初期値** $x(i) = 1.0$**としたもの。**

対称行列反復法
実行時間一覧表(秒)

プログラム	x5570	E5-2670	Phi5110P	E5-2660	E5-2660	SR16000	SR16000
	1smp	16smp	240smp	16smp	32smp	32smp	64smp
d3sor3	305.8545	107.2121	45.4492	81.5021	85.1895	18.4587	15.9792
d3sor2	5157.8583	2106.8935	845.0379	1523.1281	1658.299	285.6998	300.2200
d3adi	6484.9054	976.999	732.4666	774.8730	1005.332	291.1860	281.3099
d3bcg	137.1272	54.8386	19.8924	41.5801	38.4800	12.9068	13.1204
d3cg	90.8600	31.1266	11.2524	20.9102	19.2947	7.8468	7.7593
d3cgs	127.0035	45.2665	18.8847	37.1193	34.9020	14.0279	15.7210
d3scg	89.9558	36.7480	13.2181	24.6104	23.9617	9.1910	11.4737
d3bicgs	112.0541	31.3391	14.6020	25.8810	24.5952	10.2689	11.7728
d3cgsilu	148.3735	92.7729	572.2768	92.6816	93.9379	60.3351	61.4112
d3cgsmil	118.5756	82.1866	511.6339	81.9387	82.9746	53.7503	53.3771
d3gpbicg	122.8012	46.6471	18.2745	30.8516	33.7076	13.3188	12.8016
d3cgsm	456.1777	48.2078	43.7151	45.3634	42.8455	27.0258	27.2273
d3cgsmm	487.1845	58.6426	77.2315	73.7438	62.3460	34.3474	42.1951

E5-2670,E5-2660,Phi5110PはSR16000と比較して実行効率ではほぼ同等の性能がでています。

4.3.2 非対称問題

演算は4倍精度演算

問題は3次元ポアソン方程式

$$\Delta u + R\frac{\partial u}{\partial x} = -f$$

領域 $[0,1]\times[0,1]\times[0,1]$

解析解 $u(x,y,z) = e^{xyz}\times\sin(\pi x)\times\sin(\pi y)\times\sin(\pi z)$

非対称問題。

解法

bcg(biconjugate gradient)**法**

$R = 100, nx = ny = nz = 129$

cgs(conjugate gradient square)**法**

$R = 100, nx = ny = nz = 129$

収束判定値は共役残差 0^{-12}

初期値 $u(x,y,z) = 0$

非対称問題実行時間一覧表(秒)

プログラム	E5-2670	Phi5110P	E5-2660	E5-2660	SR16000	SR16000
	16smp	240smp	16smp	32smp	32smp	64smp
bcg	47.8155	92.7816	**75.5405**	**54.6095**	5.6664	4.1678
cgs	50.6370	99.7317	77.4960	59.7968	9.3170	7.4054

4倍精度演算ではSR16000の性能がカタログ性能比を考慮しても非常に良い性能を示しています。

今回のcgs法は前処理付きだが以前の方式ではサイズが113*113*113で

6倍精度	**SR1 6000**	**64smp**	**29.1450秒**
	E5-2670	**16smp**	**47.0610秒**
8倍精度	**SR1 6000**	**64smp**	**33.5245秒**
	E5-2670	**16smp**	**70.0802秒**

と倍精度変数をつなげた形式ではE5-2670の実行効率が勝っていました。

4.4 QCD計算

テストケース

$case1 \quad data \quad 16\times16\times16\times16$

$case2 \quad data \quad 24\times24\times24\times24$

$case3 \quad data \quad 32\times32\times32\times32$

測定条件

$BG/Q, SR16000/M1, xm1$ すべて1ノード

smt数 $BG/Q, xm1$ $smt = 1,2,4$

$SR16000/M1$ $smt = 1,2$

スレッド並列のソース

BG/Q ppgenで作成したソース。-qsmp=auto でコンパイル.

SR16000/M1,xm1 ともに自動並列 でコンパイル。

QCD 性能測定結果一覧表(対1スレッドに対する台数効果)

data	BG/Q			SR16000/M1		xm1
	smt=1	smt=2	smt=4	smt=1	smt=2	最高性能smp数
case1	13.63	15.84	11.22	18.61	14.85	32
case2	12.85	13.28	12.75	9.81	11.18	32
case3	12.33	13.03	12.88	11.09	9.48	32

最高性能

BG/Q	smt=2	=>32smp
SR16000/M1	smt=1	=>32smp
xm1		=>32smp

扱うデータがおおきくなるとSR16000/M1は大きな性能低下が見られます。

BG/Qはデータサイズによる性能低下は見られません。

BG/Q smt=1 (16smp)で台数効果13と高い。

性能比較

QCD 32*32*32*64 並列実行性能測定結果

実行時間一覧表(秒)

1 ノードsmp BG/Q ソースはParallel Program Generetor で作成

SR16000	339.15	BG/Q	2221.9
SR16000/BG実行性能比 6.55			
SR16000/BG カタログ性能比 4.79			

mpi,smp並列

SR16000		BG/Q	
node	total	node	total
1	217.4	32	56.34
4	46.37		
8	18.8		

SR16000	(対BG/Q)	
node	実行性能比	カタログ性能比
1	0.259	0.15
4	1.215	0.598
8	3.029	1.197

ソースチューニングがない場合,SR16000/M1 がBG/Qに対してカタログ性能比以上に 実行性能比がでています。

4.5 量子モンテカルロ法による物性スペクトル計算

プログラムは高エネルギー加速器研究機構，物構研岩野氏より提供いただいたのをベースにしています。

数値計算からみた場合の計算内容

G_i, A_i, B, C_i : $n \times n$行列

$A_m = C_m C_{m-1} \cdots\cdots C_1$

$A_0 = I$　（単位行列）

$B = C_L C_{L-1} \cdots\cdots C_1$

$G_m = -A_m (I + B)^{-1}$

パラメータ : n, L, β, u

$0 \le m \le L, dt = \dfrac{\beta}{L}$

C_iの絶対値の最大値，最小値　$e^{\sqrt{dt \times u}}, e^{-\sqrt{dt \times u}}$

$Tr(G_m)$ を計算し，結果のチェックは$Tr(G_L) = -1$で行います。

最初の実行結果（4倍精度）

n=40, β=10, L=112, u=5　結果は正確

n=100, β=10, L=448, u=5　結果は不正確

目標　n=100, L=448を固定

β=10,20　u=5,6,7,8,9,10での結果を10進10桁一致させる。

使用されている計算手法

行列のQDR分解。
行列VをV=QR　(Q:ユニタリ行列,R:上三角行列)に分解後,Rの対角要素をDに入れ,Rの対角要素を1にする。

扱ったケース：設定したパラメータより結果の精度が見積もりやすい。

$P = e^{\sqrt{dt \times u \times L}}, E(m) = |\mathrm{Tr}(G_m)|$

プログラム実行中に現れる数値の最大値P^2,0以外の
絶対値最小値$1/P^2$

行列やベクトル要素に現れる数値の最大値P,0以外の
絶対値最小値1/P

$E(0) = 1, E(L) = 1, m = \frac{L}{2}$で$E(\frac{L}{2}) = \frac{1}{\sqrt{P}}$で最小になるので,

演算に必要な最小ビット数$\log_2 \sqrt{P} = \frac{1}{2}\log_2 P$。

精度検証結果（10進10桁一致したもの）

6倍精度　β=10,u=5,6,7,8,9
8倍精度　β=10,u=5,6,7,8,9,10
　　β=20,u=5,6,7,8
10倍精度　β=20,u=9,u=10

4.5.1 SR16000/M1 システム

性能測定条件

4倍精度　　β=10,u=5
　　結果は10進3桁までしか一致しないが
　　他との比較のため.
6倍精度　　β=10,u=6
8倍精度　　β=20,u=8
10倍精度　β=20,u=10

チューニング手順

(1) 6倍精度,8倍精度での加減算,乗算,除算,
　　平方根計算をサブルーチン化して,
　　uinlineオプションを適用
(2) 演算量の多い行列積計算,QDR積計算部分
　　をサブルーチン化し、並列実行する。
(3) 加減算,乗算において,if文削除の
　　アルゴリズムに変更。
(4) (1)~(3)の方式を取り入れて10倍精度
　　演算ルーチンを作成。

性能測定結果

量子モンテカルロ実行時間(秒)
SR16000/M1 1ノード

4倍精度

smt on,off	実行時間
32core	9.327
64smp	11.04

10倍精度

QDR積計算	64smp	32core
並列化不可	733	783
分割並列化	426	456
数値制限解除	451	435

uinlineオプション

精度	サブルーチン化	64smp	32core
6倍精度	612	263	243
8倍精度	1190	463	386

	行列積,QDR積計算		IF文削除	
精度	64smp	32core	64smp	32core
6倍精度	112	105	23.743	28.837
8倍精度	192	174	42.074	51.739

10倍精度演算の問題

加減算,乗算での演算数の増加

(1) 最適化,uinlineの効果が減少

(2) 行列積計算での並列化効果の減少

(3) QDR積計算が並列化不可となり,2つのサブルーチンに分割した事による並列化効果の減少

表現できる数値範囲の制限

大きな数の平方根計算では、アルゴリズムの変更が必要。

4.5.2 ieee754-2008形式

E5-2670	16smp	Phi5110P	240core		
モンテカルロシュミレーション					
				E5-2670	Phi5110P
精度	β	U	L	実行時間	実行時間
				(秒)	(秒)
6倍精度	10	6	448	127.5865	1088.6992
8倍精度	20	8	448	261.7628	1762.0854
10倍精度	20	10	448	619.8694	3727.0650
ieee8倍精度	20	10	448	505.0661	4487.2604
DQ(8倍精度)	20	9	448	1961.2144	16287.9011

(1) β=20,U=10を計算する場合dd形式では
10倍精度演算が必要でieee形式では
8倍精度で事足ります。
(2) SR16000でscopeオプションで実行すると
10倍精度は649秒.このため,計算機種の差
ではなく,コンパイラの最適化能力の差
が効いていると言えます。

DD形式8倍精度とieee754-2008形式の
8倍精度演算では,正しく計算出来る
パラメータ領域に差がある。

一概にどちらが良いとは言えない

β=20,U=8ではDD形式8倍精度が良い.
β=20,U=10ではDD形式では10倍精度を
使用する必要があり,ieee754-2008形式
の8倍精度が良い。

10倍精度演算ではieee754-2008形式の方
が性能が良いと考えられ,GPU等の
アクセラレータで8倍精度演算までのDD形式を
適用している理由とも合う結果となっている。

4.6 ファインマンループ積分

ファインマンループ積分の一般式は以下の様になる。

$$I(\varepsilon)=(-1)^{N}\left(\frac{1}{4\pi}\right)^{nL/2}\Gamma(N-nL/2)\int_{0}^{1}\prod_{i=1}^{N}dx_{i}\,\delta(1-x_{1}-x_{2}-.....-x_{N})\frac{C^{N-n(L+1)/2}}{(D-i\varepsilon C)^{N-nL/2}}$$

n : **時空次元** $(n=4)$, N : **ループ内の素粒子の数**, L : **ループの多重度**

ε : **実定数**, C, D : $x_1, x_2, ..., x_N$ **の多項式**

4.6.1 inf ra vtx 計算

$$I=\int_{0}^{1}\int_{0}^{1-x}\frac{1}{D}dydx$$

$$D=-xys+(x+y)^{2}m^{2}+(1-x-y)\lambda^{2}$$

$$s=-500^{2},\ m=0.0005$$

倍精度演算で $\lambda=10^{-5}$ **で実施しています。**

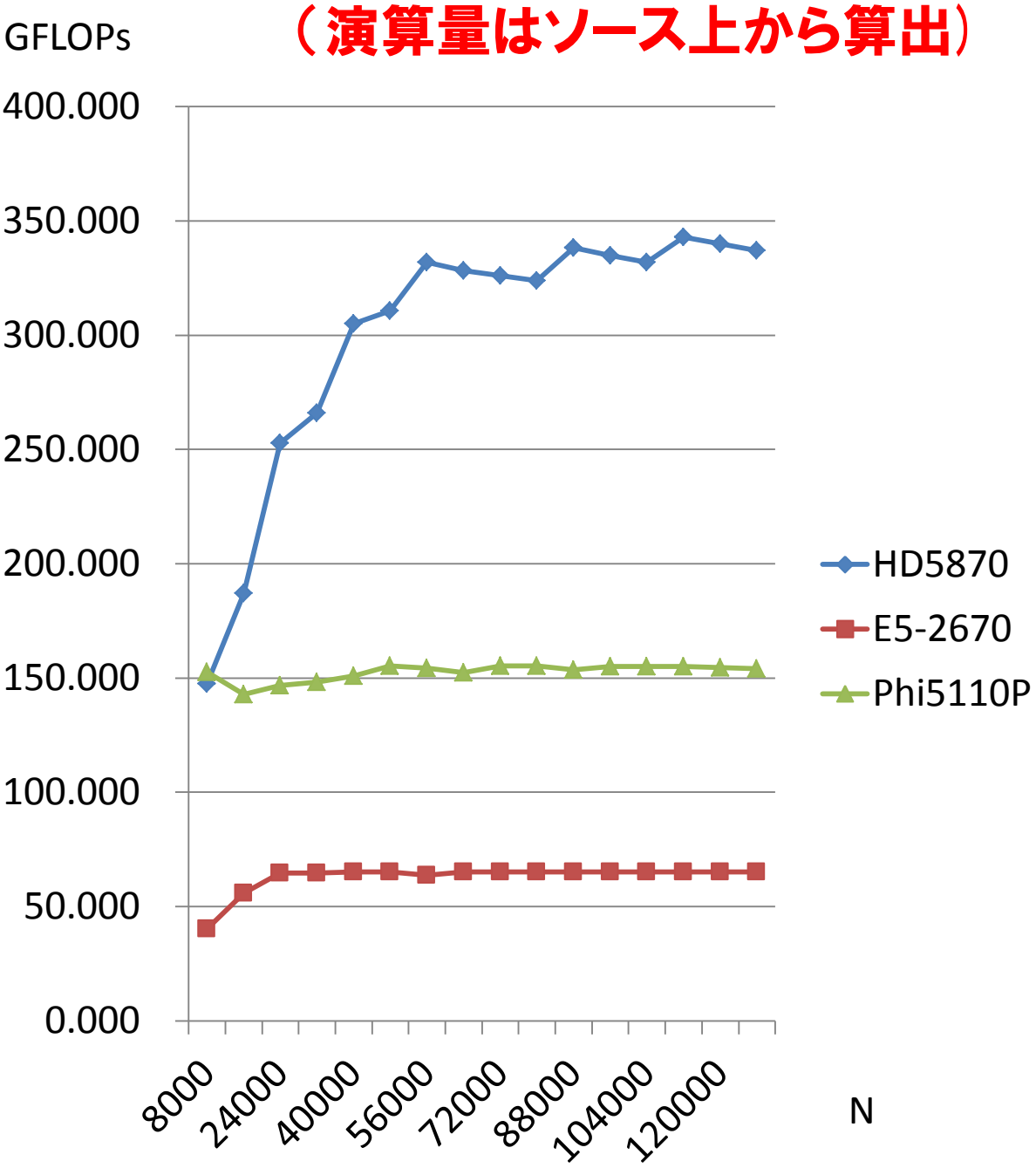
実行条件

HD5870　3200SMP

E5－2670　16SMP

Phi5110P　240SMP

2次元積分(VTX)性能測定結果
(演算量はソース上から算出)



サイズ

4.6.2 Infra box 計算

計算式

$$I = \int_0^1 \int_0^{1-x} \int_0^{1-x-y} \frac{1}{D^2} dz dy dx$$

$$D = -sxy - tz(1-x-y-z) + (x+y)\lambda^2 + (1-x-y-z)(1-x-y){m_e}^2 + z(1-x-y){m_f}^2$$

テストデータ

$$s = -500^2, t = -150^2, m_f = 150,$$

$$m_e = 0.0005$$

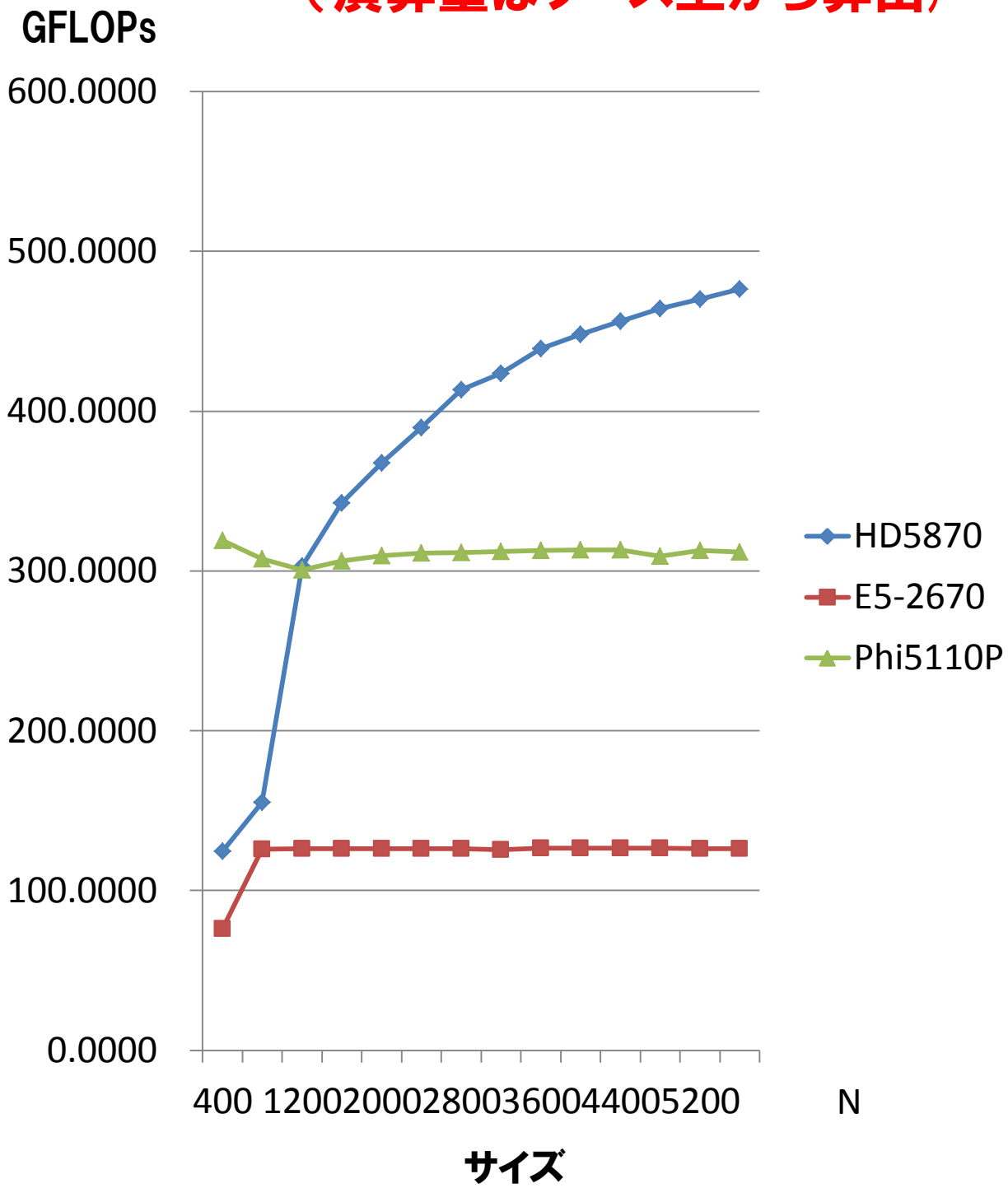
$$\lambda = 10^{-12}(\text{倍精度}), \lambda = 10^{-30}(\text{4倍精度})$$

倍精度

N=8192,演算量10447GFLOP

SR16000/M1	1ノード	自動並列		
BG/Q	32ノード	ノード間	MPI	
		ノード内	スレッド並列	
計算機	smp数	smt数	性能(GFLOPs)	実行効率(%)
SR16000/M1	32	1	268	27
SR16000/M1	64	2	263	27
BG/Q	16	1	846	13
BG/Q	32	2	1493	23
BG/Q	64	4	1766	27
HD5870	1	3200	326	30
HD6970	1	6144	588	22
	実行効率は25%－30%			

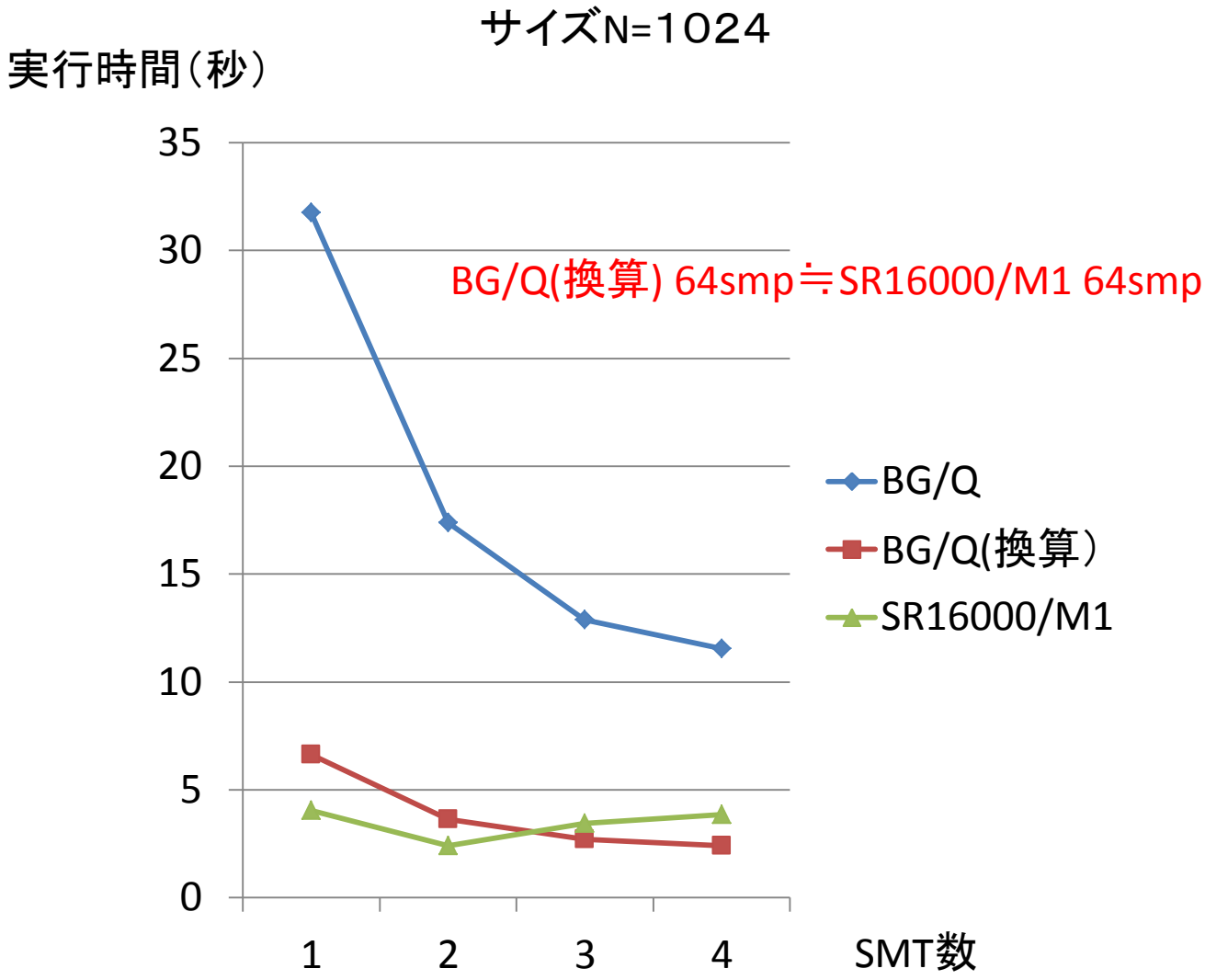
3次元積分（BOX）性能測定結果
（演算量はソース上から算出）



4倍精度

サイズ	N=2048	演算量	618GFLOP		
テーブルサイズ	32KB				
SR16000/M1	4,8ノード	フラットMPI			
BG/Q	32ノード	フラットMPI			
	4倍精度性能測定結果(GFLOPs)				
計算機	ノード数	smt数	性能	実行効率(%)	
SR16000/M1	4	1	248	6.3	
SR16000/M1	4	2	418	10.7	
SR16000/M1	8	1	495	6.3	
SR16000/M1	8	2	836	10.7	
BG/Q	32	1	253	3.9	
BG/Q	32	2	475	7.2	
BG/Q	32	4	705	10.8	
HD5870	2ボード	3200	283	26	
HD6970	4ボード	6144	560	20.7	
	SR16000/M1,BG/QではSIMDが適用されない				
	ので,GPGPU系の計算機の実行効率が良い				

１ノード性能比較

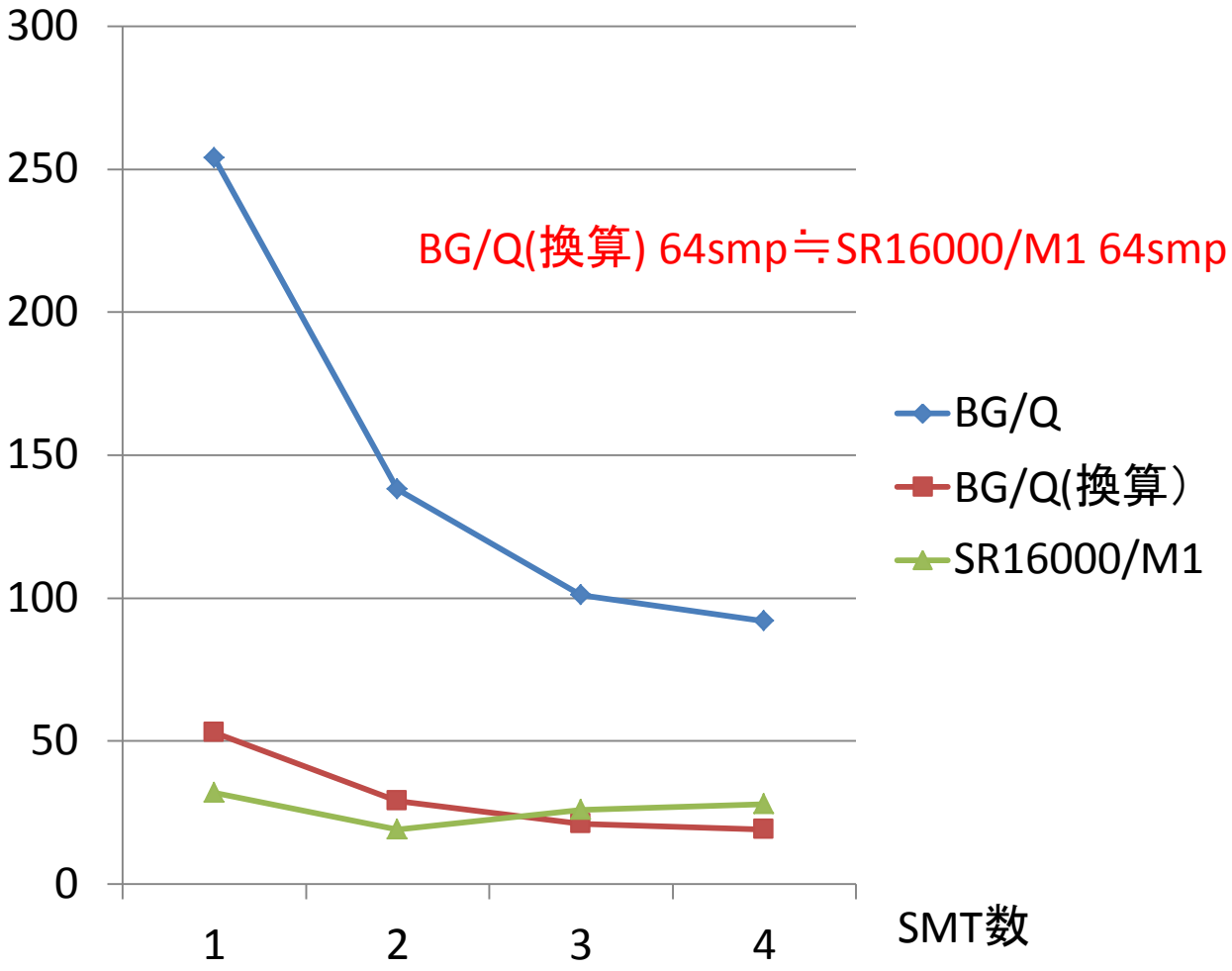


	BG/Q	SR16000/M1
SMT=1	16 smp	32 smp
=2	32 smp	64 smp
=3	48 smp	96 smp
=4	64 smp	128 smp

BG/Q(換算)　204.8GFLOPs => 980.48GFLOPs にしたときの値

サイズN=2048

実行時間（秒）

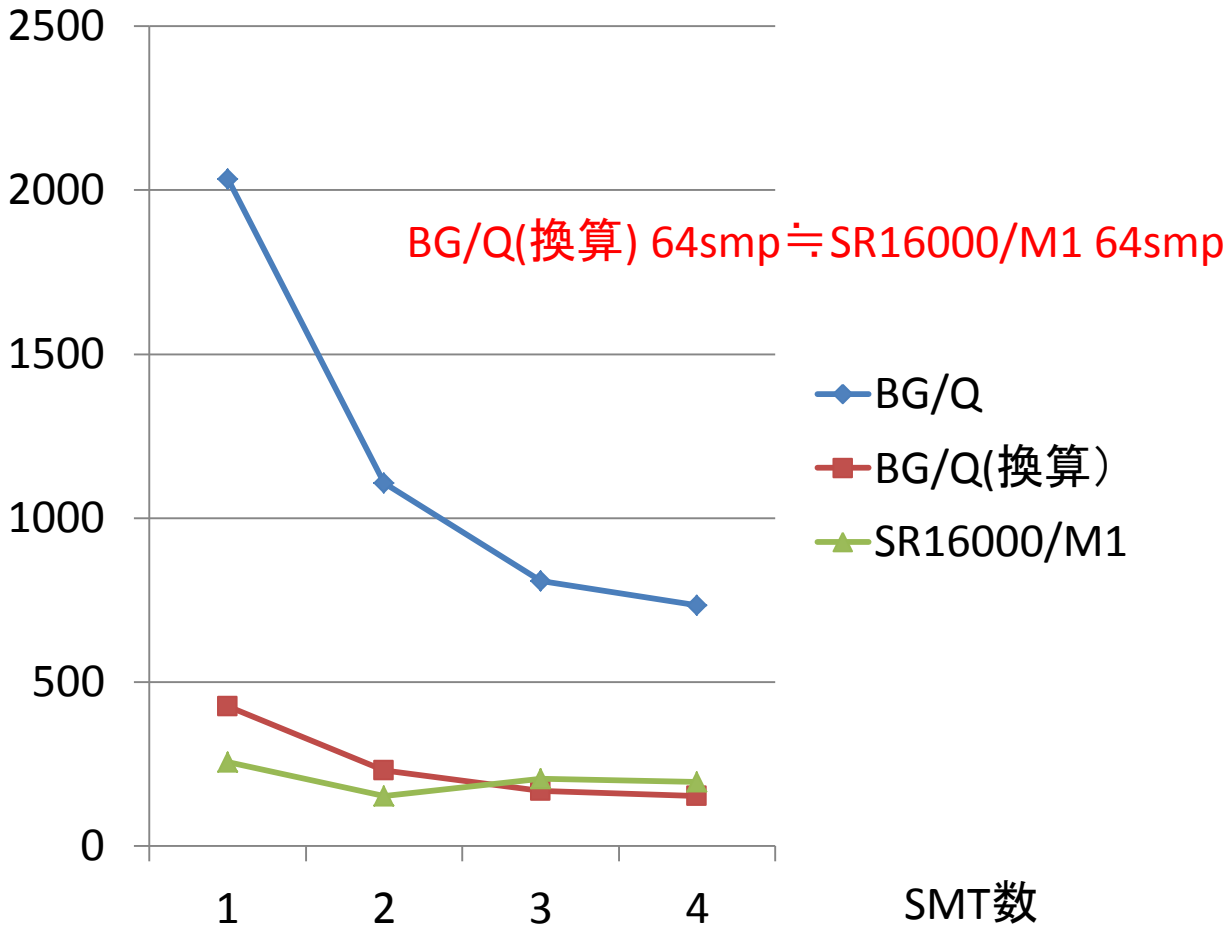


	BG/Q	SR16000/M1
SMT=1	16 smp	32 smp
=2	32 smp	64 smp
=3	48 smp	96 smp
=4	64 smp	128 smp

BG/Q(換算)　204.8GFLOPs => 980.48GFLOPs にしたときの値

サイズN=4096

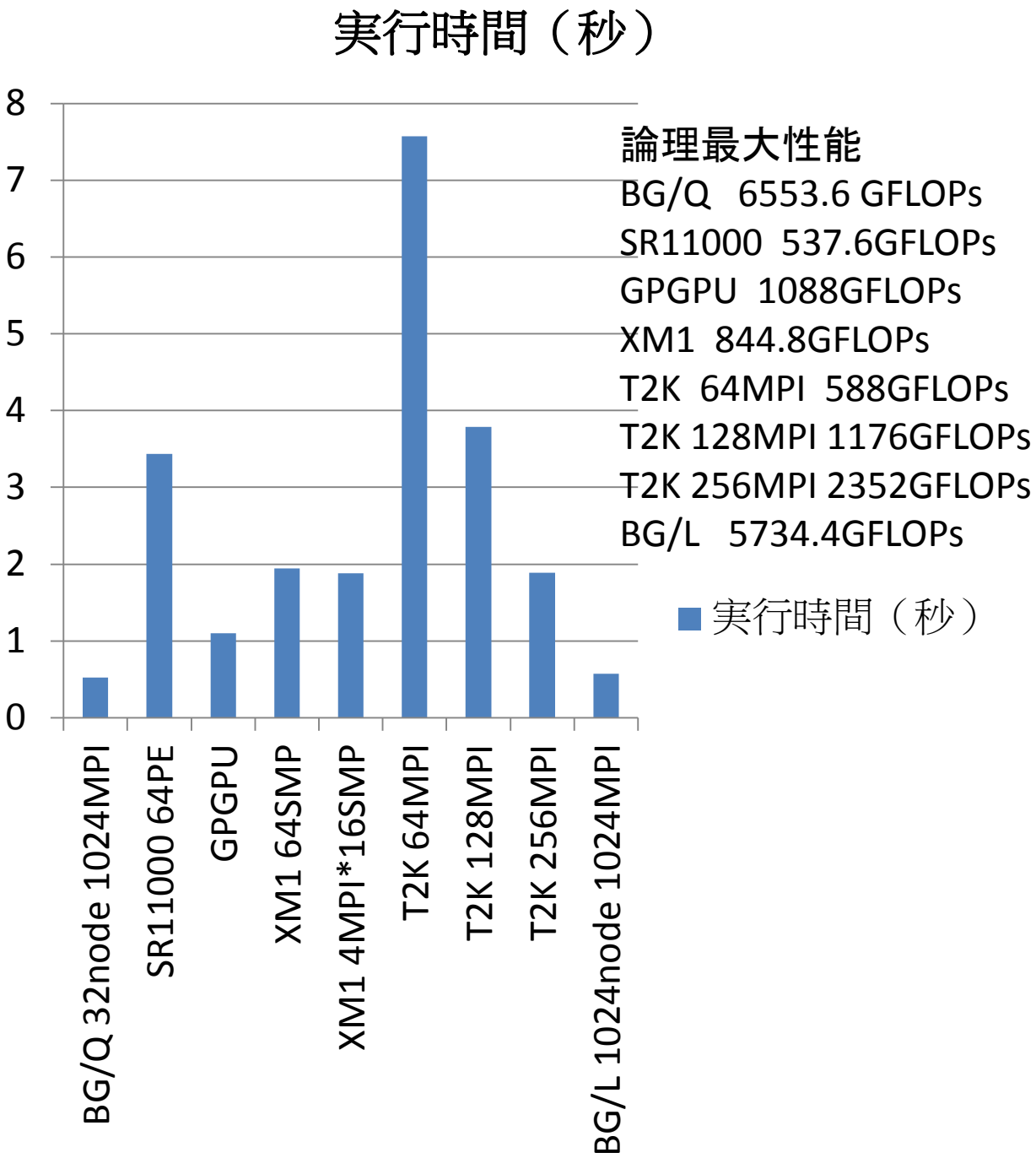
実行時間（秒）



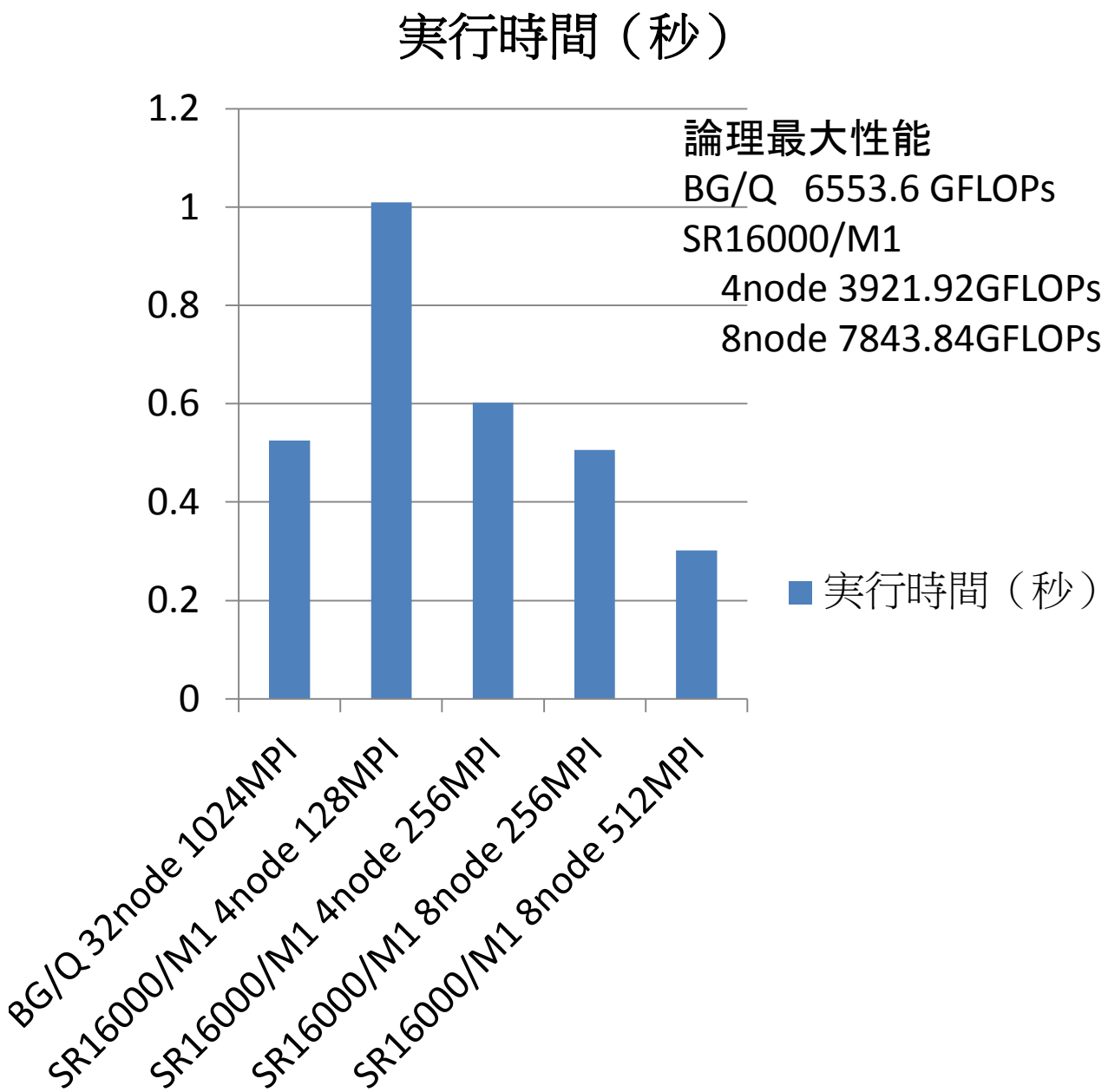
	BG/Q	SR16000/M1
SMT=1	16 smp	32 smp
=2	32 smp	64 smp
=3	48 smp	96 smp
=4	64 smp	128 smp

BG/Q(換算)　204.8GFLOPs => 980.48GFLOPs にしたときの値

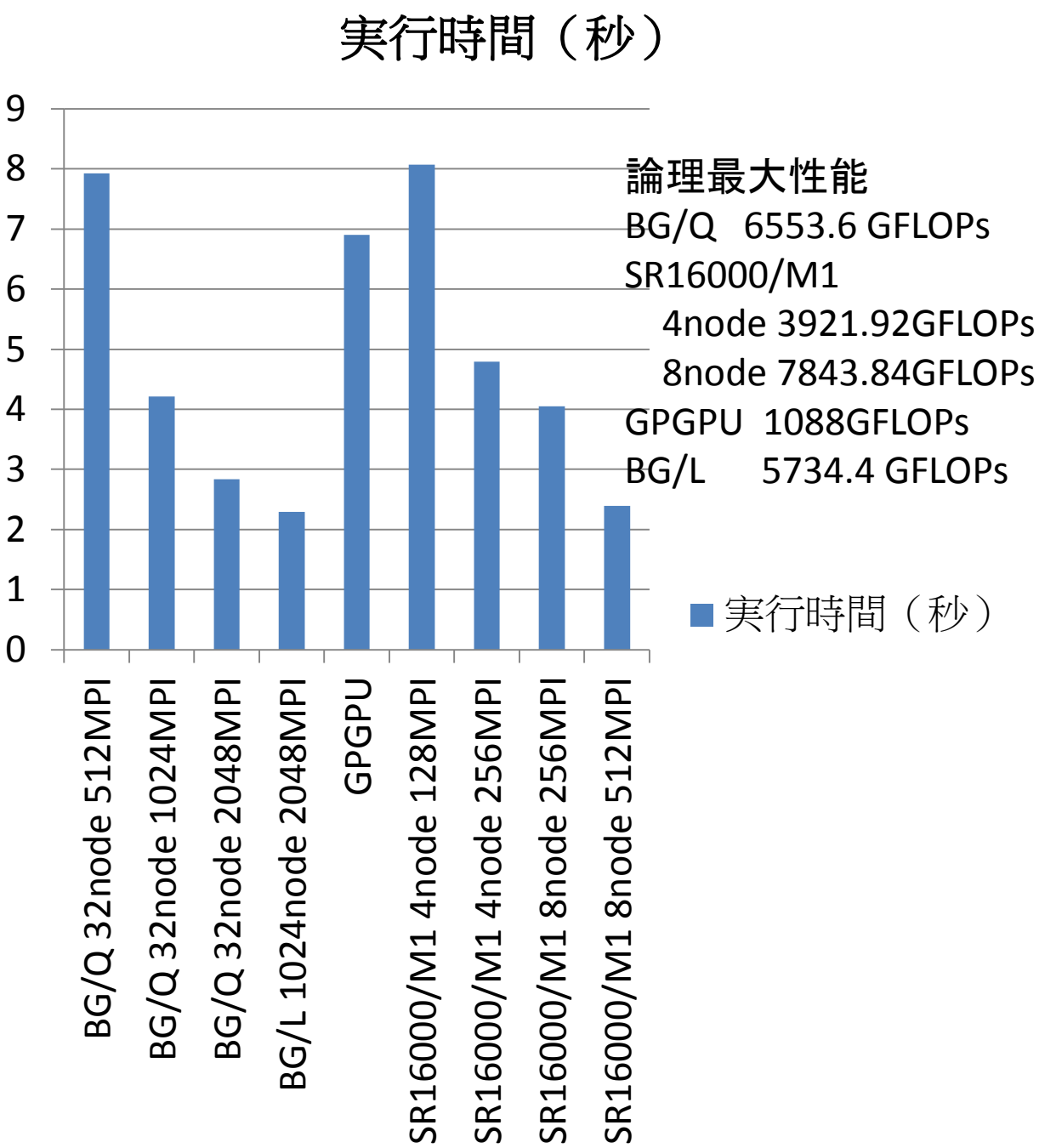
4倍精度INFRA BOX N=1024 各種計算機実行時間(1)



4倍精度INFRA BOX N=1024 各種計算機実行時間(2)

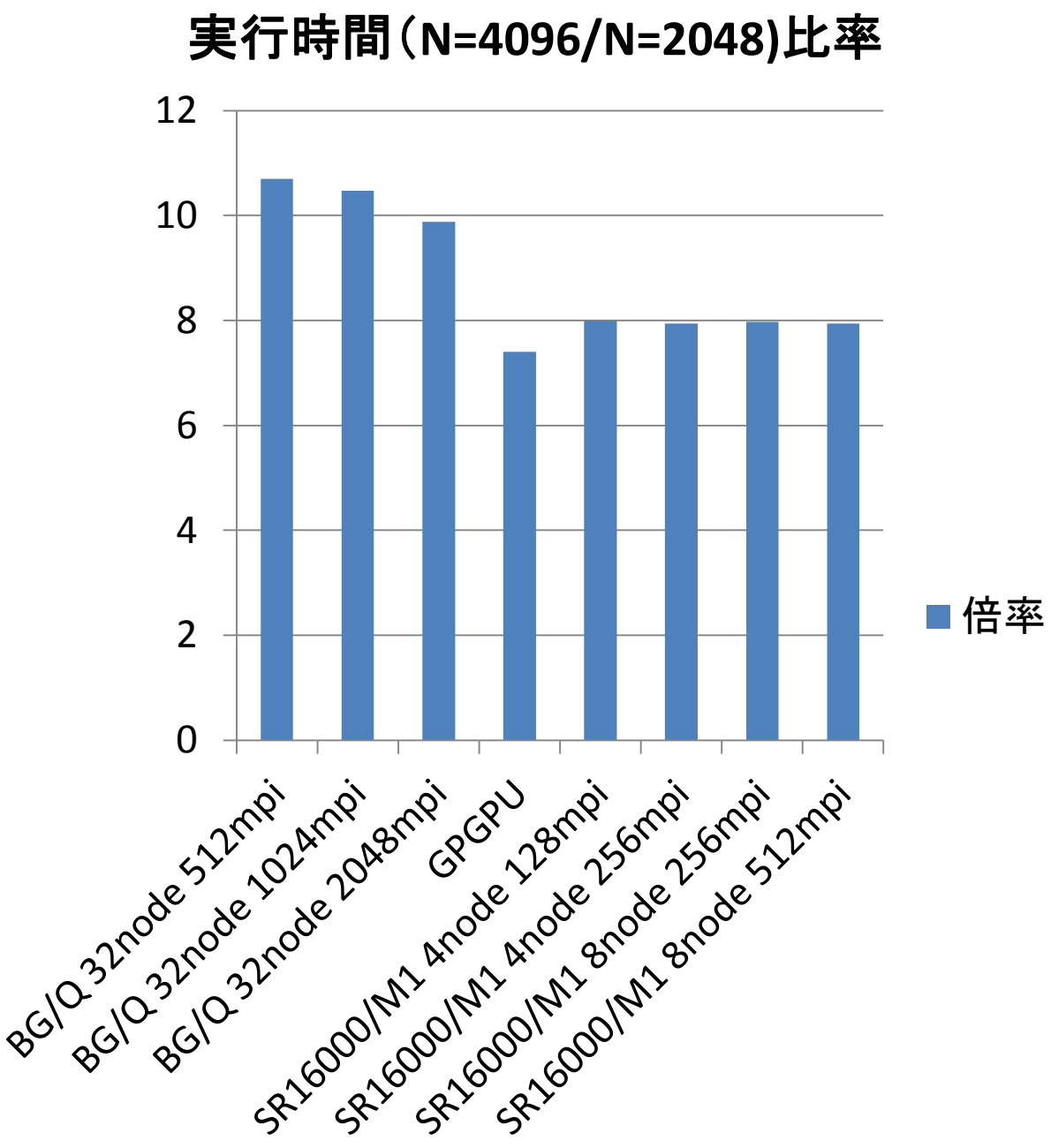


4倍精度INFRA BOX N=2048 各種計算機実行時間



4倍精度INFRA BOX N=2048とN=4096
各種計算機実行時間比率

演算量比率(N=4096/N=2048)=8



4.6.3 massless計算

$$\eta^n \times \int_0^1 \int_0^{1-x} \cdots\cdots \int_0^{1-x_1-\cdots\cdots x_{n-1}} \frac{1}{(x_1 x_2 \cdots\cdots x_n)^{1-\eta}} dx_n dx_{n-1} \cdots\cdots dx_1$$

$$= (1+n\eta)(2+n\eta)\cdots\cdots(n+n\eta) \times$$

$$\int_0^1 \int_0^{1-x} \cdots\cdots \int_0^{1-x_1-\cdots\cdots x_{n-1}} (x_1 x_2 \cdots\cdots x_n)^{\eta} dx_n dx_{n-1} \cdots\cdots dx_1$$

となり，倍精度演算で精度良く計算するには，

$$(1+n\eta)(2+n\eta)\cdots\cdots(n+n\eta) \times$$

$$\int_0^1 \int_0^{1-x} \cdots\cdots \int_0^{1-x_1-\cdots\cdots x_{n-1}} (x_1 x_2 \cdots\cdots x_n)^{\eta} dx_n dx_{n-1} \cdots\cdots dx_1$$

を計算する必要があります。

倍精度演算による実行結果

条件一覧表

n	h	N
3	0.5^6=0.015625	503
4	0.5^6=0.015625	503
5	0.5^4=0.0625	103
6	0.5^3=0.125	51
7	0.16	41
8	0.20	33

実行時間一覧表(秒)

smt=on,64smp　　#NAME?

n	SR16000	xm1
3	0.148960	0.278902
4	74.977050	112.659721
5	16.623410	23.287427
6	25.476964	38.277622
7	362.011287	417.984915
8	3199.456509	3635.130558

n	Phi5110P 240smp	E5-2670 16smp
3	0.6069	0.1777
4	29.1766	81.6845
5	47.3931	15.6406
6	20.8052	27.8814
7	1478.4449	265.0752
8	13738.5545	2255.2062

Phi5110Pでのチューニング例

$$\int_0^1 \int_0^{1-x} \cdots\cdots \int_0^{1-x_1-\cdots\cdots x_{n-1}} \frac{1}{(x_1 x_2 \cdots\cdots x_n)^{1-\eta}} dx_n dx_{n-1} \cdots\cdots dx_1$$

$$= \frac{1}{\eta^n} \frac{(\Gamma(1+\eta))^n}{\Gamma(1+n\eta)}$$

$$\eta^n \int_0^1 \int_0^{1-x} \cdots\cdots \int_0^{1-x_1-\cdots\cdots x_{n-1}} \frac{1}{(x_1 x_2 \cdots\cdots x_n)^{1-\eta}} dx_n dx_{n-1} \cdots\cdots dx_1$$

$$= (1+n\eta)(2+n\eta)\cdots(n+n\eta) \int_0^1 \int_0^{1-x} \cdots\cdots \int_0^{1-x_1-\cdots\cdots x_{n-1}} (x_1 x_2 \cdots\cdots x_n)^{\eta} dx_n dx_{n-1} \cdots\cdots dx_1$$

において$\eta = 0.01, n = 7, n = 8$の場合を計算した。

倍精度で計算するため，右辺の式を使用。

$n = 7, h = 0.16, N = 41$

$n = 8, h = 0.2, N = 33$

で実行した結果は下記 のとうり。

n=7,n=8 massless計算実行時間(秒)一覧表				
SR16000 64smp,-Os 自動並列				
XM1　64smp,-Os 自動並列				
E5-2670 16smp -Os -openmp				
Phi5110P 240smp -Os -openmp				
n	SR16000	XM1	E5-2670	Phi5110P
6	25	38	21	27
7	362	418	265	1478
8	3199	3635	2255	13739

n=7,n=8の場合のPhi5110Pの値が悪すぎる。

n = 7　　実行時間　1478秒 => 368秒

n = 8　　実行時間　13739秒 => 2623秒

n = 7だとSR16000とほぼ同等, n = 8だとSR16000の1.2倍の性能。

結果

n = 7　解析解　0.996679645317184

実測結果　0.996679645317182

n = 8　解析解　0.9955954352742048

実測結果　0.995595435126120

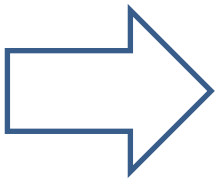
N = 41の場合の

実測結果　0.995595435274190

結果は倍精度演算でも十分な精度となっている。
n=7とn=8の差はh=0.16 (N=41) ,h=0.2 (N=33) の違いによる。
一般にDEでは倍精度演算ではN=40-50で十分な精度が得られる。

このためx,yを以下の様に一次元化

```
do  i1=1,n
xx=x30(i1)
by=1.0d0-xx
Do i2=1,n
yy=x30(i2)*by
bz=1.0d0-xx-yy
```



```
do i1=1,n*n
xx=xlist(i1)
yy=ylist(i1)
bz=1.0d-xx-yy
```

gw30(i1)*by*gw30(i2)

wlist(i1)

前処理

```
l=0
do ip=1,n
x1=x30(ip)
b2=1.0d0-x1
do iq=1,n
x2=x30(iq)*b2
l=l+1
xlist(l)=x1
ylist(i)=x2
wlist(l)=gw30(ip)*b2*gw30(iq)
end do
end do
```

4.6.4 4-6次元積分計算
4.6.4.1 S221計算

$$S^{221}(s;m_1^2,m_2^2,m_3^2,m_4^2,m_5^2)=\int_0^1\int_0^{1-x}\int_0^{1-x-y}\int_0^{1-x-y-z}\frac{1}{DC}dudzdydx$$

$$C=(x+y+z+u)(1-x-y-z-u)+(x+y)(z+u)$$
$$E=(1-x-y-z-u)(x+z)(y+u)+(x+y)zu+(z+u)xy$$
$$M^2=xm_1^2+ym_2^2+zm_3^2+um_4^2+(1-x-y-z-u)m_5^2$$
$$D=-sE+M^2C$$

$$s^{221}(-1:100,100,0,0,100)$$

解析近似解：0.038000443812 7

	サイズ N=2048,演算量=281622GFLOP					
	テーブルサイズ 32KB					
	SR16000/M1	1ノード	BG/Q 32ノード			
	計算機	SR16000/M1	SR16000/M1	BG/Q	BG/Q	
	並列化	smt=1	smt=2	flat	smp	
	実行時間(秒)	1872	1187	244	207	
	性能(GFLOPs)	150	237	1054	1360	

$$S^{221}(5;100,100,0,0,100)$$

$$\varepsilon = 1.2^{-15}$$

$$N = 1024$$

		GPU
HD5870	122.551	sec
HD6970	55.667	sec
		HOST
HD5870	79903.86	sec
HD6970	96984.31	sec

$S^{221}(-1;100,100,0,0,100)$の性能測定結果

サイズ N=576,演算量 1762GFLOP (性能モニター)

X5570,SR11000 自動並列,E5430 MPI

スレッド数	X5570		E5430		SR11000	
	実行時間	性能	実行時間	性能	実行時間	性能
	秒	GFLOPs	秒	GFLOPs	秒	GFLOPs
1	367.441	4.795	460.579	3.826	1477.16	1.193
2	204.562	8.614	232.52	7.578	738.884	2.385
4	98.036	17.973	116.234	15.159	369.904	4.763
8	48.402	36.403	90.206	19.533	185.167	9.516
16	57.326	30.736	58.128	30.312	92.873	18.972
32			28.79	61.202		
				MPI16	175.644	10.032
				MPI32	87.315	20.18
				MPI64	43.782	40.245

BG/L MPI

スレッド数	N	演算量	実行時間	性能
		GFLOP	秒	GFLOPs
256	1024	17601	273.879	64.266
512	1024	17601	136.939	128.532
1024	1024	17601	68.469	257.065
2048	2048	281622	557.638	505.027

$S^{221}(1;100,100,0,100)$の性能測定結果

ε – 算法 反復回数 15回

CPU	N	演算量	スレッド数	実行時間	性能
		GFLOP		秒	GFLOPs
BG/L	1024	264015	1024	1061	249
X5570	548	21660	8	609	36
E5430	576	26430	32	516	51

$s^{221}(5;100,100,0,0,100)$ $\varepsilon = 1.2^{-15}$ $N = 1024$

演算量 17601GFLOP

実行時間(秒)一覧表

ボード数	HD5870		HD6970
	gpu0	gpu1	
1	242.313	243.377	221.059
2	122.455		110.715
3			74.519
4			55.638

HD5870 143GFLOPs,HD6970 316GFLOPs SR16000/M1 システムに匹敵する性能がでています。

		GPU
HD5870	122.551	sec
HD6970	55.667	sec
		HOST
HD5870	79903.86	sec
HD6970	96984.31	sec

GPU がHOSTに比べて非常に良い性能がでています

HOSTコンピュータでの性能が低い事の原因追究

(1) GPU用に4重DOループを2重DOループにしたのが原因　?(4次元積分,2次元積分)

(2) 使用コンパイラの問題では？

実行時間一覧表(秒)

cpu	コンパイラ	4次元積分	2次元積分	
x5570	gcc	60808	57308	
x5570	ifort	4227	6255	
cpu	コンパイラ	4次元積分	2次元積分	
e5430	icc	5918	12616	
e5430	icc parallel	928	2785	8smp

ソースで二次元化したためキャシュミスが発生。
コンパイラの最適化処理能力。

GPUでのチューニングは、容量の大きいベクトルレジスタを持ったベクトル機でのチューニングと同じ傾向を示します。

スーパーコンピュータでの4次元積分と2次元積分の比較

実行時間一覧表（秒）		64MPI
CPU	4次元積分	2次元積分
xm1	15.00185	25.03721
SR16000/M1	8.80442	17.75754
T2K	11.88899	31.80749

4次元積分の方が性能が良いという傾向はサーバー系x5570,e5430と同じ。

f90とxlf_r コンパイラの比較

$S^{221}(-1;100,100,0,0,100)$　　$N=1152$

演算量 $=47505$　　$GFLOP$

SR16000/M1,xm1　1ノード

cpu	smp数	f90	xlf_r
xm1	32	247.354	133.49
xm1	64	279.959	131.89
xm1	96	274.067	133.65
xm1	128	205.521	130.01
SR16000/M1	64	178.875	
SR16000/M1	32	147.387	

f90　xm1　実行効率27.4%,
SR16000/M1　32.9%
と非常に良い値がでています。
xlf_r　はf90以上に実行効率43．3%とよく,
キャシュの有効利用とSIMDが非常によく効いています。

SR16000,E5-2670,Phi5110Pの比較の結果は以下の様になっています。

4次元積分実行時間一覧表(秒)

case	精度	SR16000		E5-2670	Phi5110P	演算量
		32core	64smp			((GFLOP)
case1	倍精度	11.698903	11.645035	27.618532	17.836520	2976
case1	拡張倍精度			108.657508	257.571165	
case2	倍精度	24.126208	30.395169	26.958704	17.815714	4568
case2	拡張倍精度			108.635537	256.691088	
case2	4倍精度			4538.965000	9762.636500	

(1) 多重DOループの一重化はE5-2670, Phi5110Pは若干速くなりますがSR16000は2~3倍程度遅くなっています。
(2) ieee754-2008形式の4倍精度は非常に遅くなっています。

サイズを大きくした場合とHD5870との比較は以下の様になっています。

4次元積分計算詳細テスト結果 (s221)				
case2	N=1200	実行時間(秒)		
精度	E5-2670	Phi5110P		
	16smp	240core		
倍精度	506.127663	287.737872		
拡張倍精度	2047.347488	4682.559193		
case2		実行時間(秒)		
精度	HD5870		Phi5110P	
	1面	2面	192core	240core
倍精度	27.190329	13.814706	19.255160	17.815714
拡張倍精度			302.454400	256.691088
4倍精度	453.623999	232.079049		

Phi5110Pは倍精度は高速でE5-2670は拡張倍精度が高速。HD5870の4倍精度はdd形式のため高速です。

S^{221} テスト

$S^{221}(-1,100,100,0,0,100)$

$N = 576$

演算量	GFLOP
倍精度	4568
4倍精度	42187

SR16000　実行時間(秒)

精度	32core	64smp
倍精度	30.395169	24.126208
4倍精度	550.758631	330.429147

E5-2670　16SMP

精度	実行時間(秒)
倍精度	26.9587
拡張倍精度	108.6355
4倍精度	4008.4746

HD5870　実行時間(秒)

精度	1面	2面
倍精度	27.1903	13.8147
4倍精度	453.6240	232.0790

Phi5110P　実行時間(秒)

精度	1smp /core	2 smp/core	3smp/core	4smp/core
倍精度	42.0126	27.6116	23.3237	20.6132
拡張倍精度	335.2602	258.8425	255.4137	249.6453
4倍精度	3364.0728	1822.1579	1683.9084	1488.7508

HD6970 実行時間(秒)

精度	一面	二面	三面	四面
倍精度	11.443008	5.923182	4.036383	3.196938
4倍精度	322.827294	162.131639	107.290817	82.046807

グラフィックボード性能比較表

精度	HD7970	W8000	HD7980
倍精度	9.164966	9.455362	9.963973
4倍精度	267.937949	305.351604	304.181516

4.6.4.2　5次元積分計算

laportad

$$I=\int_0^1\int_0^{1-x_1}\int_0^{1-x_1-x_2}\int_0^{1-x_1-x_2-x_3}\int_0^{1-x_1-x_2-x_3-x_4}\frac{1}{D^2}dx_5dx_4dx_3dx_2dx_1$$

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

```
D =
    &-x1**2*x2-x1**2*x3-x1**2*x4-x1**2*x6-x1*x2**2-x1*x2*x3
    &-2.d0*x1*x2*x4
    &-x1*x2*x5-x1*x2*x6-x1*x3**2-2.d0*x1*x3*x4-x1*x3*x5-
x1*x3*x6
    &-x1*x4**2
    &-x1*x4*x5-2.d0*x1*x4*x6-x1*x5*x6-x1*x6**2- x2**2*x4-
x2**2*x5
    &-x2*x3*x4
    &-x2*x3*x5-x2*x4**2-2.d0*x2*x4*x5-x2*x4*x6-x2*x5**2-
x2*x5*x6
    &-x3**2 *x4
    &-x3**2*x5-x3*x4**2-2.d0*x3*x4*x5-x3*x4*x6-x3*x5**2-
x3*x5*x6
    &-x4**2*x5
    &-x4**2*x6-x4*x5**2-3.d0*x4*x5*x6-x4*x6**2-x5**2*x6-
x5*x6**2
```

解析近似解=0.2762092253588

laportaf

$$I=\int_0^1\int_0^{1-x_1}\int_0^{1-x_1-x_2}\int_0^{1-x_1-x_2-x_3}\int_0^{1-x_1-x_2-x_3-x_4}\frac{1}{D^2}dx_5dx_4dx_3dx_2dx_1$$

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

D=

```
    .-x1*x1*x2-x1*x1*x4-x1*x1*x5-x1*x1*x6-x1*x2*x2-x1*x2*x3
    .-x1*x2*x4
    .-2.0d0*x1*x2*x5-2.0d0*x1*x2*x6-x1*x3*x4-x1*x3*x5-x1*x3*x6
    .-x1*x4*x4-3.0d0*x1*x4*x5
    .-2.0d0*x1*x4*x6-x1*x5*x5-x1*x5*x6- x1*x6*x6-x2*x2*x3
    .-x2*x2*x5-x2*x2*x6
    .-x2*x3*x3-x2*x3*x4-x2*x3*x5-2.0d0*x2*x3*x6-x2*x4*x5
    .-x2*x4*x6-x2*x5*x5-x2*x5*x6-x2*x6*x6-x3*x3*x4-x3*x3*x5
    .-x3*x3*x6-x3*x4*x4-2.0d0*x3*x4*x5-2.0d0*x3*x4*x6-
x3*x5*x5-x3*x5*x6
    .-x3*x6*x6-x4*x4*x5-x4*x4*x6-x4*x5*x5-x4*x5*x6-x4*x6*x6
```

解析解は算出できなったため,2つの異なる積分方法で結果を確認。

Laporta （d）

SR16000/M1,xm1　1ノード。
サイズN＝128、演算量　2668GFLOP

実行時間（秒）一覧表				
smp数	xm1	xm1	sr16000/M1	SR16000/M1
	f90	xlf_r	f90	xlf_r
32	30.005	52.86	11.615	38.46
64	31.151	54.52	7.498	38.51
128	19.369	54.08		

f90,SR16000/M1　の実行効率36.2%と
突出して良い値となっています。

laporta　(f)

サイズN＝128、演算量 3642GFLOP
ソースは二次元積分

スーパーコンピュータとGPU

	実行時間(秒)					
CPU	ノード数	smp数	mpi数	実行時間	性能	実行効率
				(秒)	(GFLOPs)	(%)
BG/Q	32	1	2048	1.289	2825	43.1
SR16000	1	64	1	8.808	413	42.2
HD5870	2	3200	1	8.094	450	41.4
HD6970	4	6144	1	4.252	857	31.7

どの機種も高い実行効率をしめしています。

GPUとHOST

	実行時間(秒)		
cpu	gpu	host	host/gpu
HD5870	8.094	2459	304
HD6970	4.252	3009	708

各種計算機詳細

GPU

実行時間(秒)一覧表

ボード数	HD5870		HD6970
	gpu0	gpu1	
1	15.678	15.702	15.428
2	8.08		7.894
3			5.448
4			4.251

BG/Q

case1 -O5 -qhot=level=2
case2 -O3 -qstrict -qhot=level=1

32ノード実行時間(秒)

case	flat	hybrid
1	1.289	1.324
2	4.855	4.106

GPU:ボード数の効果がよくでています。
BG/Q: コンパイルオプションの効果が大きい.
オプションによりflat MPIとハイブリッドMPIの効果が変わっています。

SR16000/M1

1ノード実行時間(秒)

smt	smp	実行時間(秒)	性能(GFLOPs)	実行効率(%)
on	64	7.909	460	47
off	32	11.674	312	31.8

T2K

実行時間(秒)一覧表

MPI数	実行時間(秒)	性能(GFLOPs)	実行効率(%)
64	17.275	211	35.6
128	8.646	421	35.8
256	4.33	841	35.8
512	2.199	1656	35.2

XM1

実行時間(秒)一覧表　論理コア数64

f90	smt	32	64
	on	88.458	55.063
	off	57.756	
xlf_r	execution	29.01	18.95

実行時間(秒)一覧表　論理コア数 128

smp	32	64	128
f90	40.911	38.31	24.677
xlf_r	26.62	22.22	20.82

SR16000/M1,T2K ともに実行効率は高くなっています。

Xm1: 論理コア数増加の効果はf90に大きくでています.

演算量　(N=120)

case	GFLOP
laporta d	2514
laporta f	2638

実行時間一覧表(秒)

CPU	SR16000		E5-2670		Phi5110P		HD5870
プログラム	32core	64smp	c	fortran	c	fortran	2面
laporta d	8.404	5.426	27.756	27.621	7.474	7.665	5.868
laporta f	9.841	6.373	28.275	27.921	7.324	7.462	5.928

演算量が約2500GFLOP程度だとSR16000とPhi5110P,HD5870の性能はほぼ等しい.
これは4次元積分でも同様でした。

4.6.4.3　6次元積分計算

laportag

$$I=\int_0^1\int_0^{1-x_1}\int_0^{1-x_1-x_2}\int_0^{1-x_1-x_2-x_3}\int_0^{1-x_1-x_2-x_3-x_5}\int_0^{1-x_1-x_2-x_3-x_5-x_6}\frac{C}{D^3}dx_7dx_6dx_5dx_3dx_2dx_1$$

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

```
C=x1*x4+x1*x5+x1*x6+x2*x4+x2*x5+x2*x6+x3*x4+x3*x5+x3*x6+x
4*x5
    &   + x4*x6+x4*x7+x5*x7+x6*x7

    D =
    &-
(x1**2+x2**2+x3**2+x7**2+x1*x2+x1*x3+x1*x7+x2*x3+x2*x7+x3
*x7)
    &*(x4+x5+x6)
    &-x4**2*(x1+x2+x3+x5+X6+x7)
    &-(x5**2+x6**2+x5*x6)*(x1+x2+x3+x4+x7)
    &-3.0*x4*(x1*x5+x6*x7)
    &-2.0*((x1+x2+x3)*x4*x6+(x2+x3+x7)*x4*x5)
```

解析近似解=0.1723367907503

laportah

$$I=\int_0^1\int_0^{1-x_1}\int_0^{1-x_1-x_3}\int_0^{1-x_1-x_3-x_2}\int_0^{1-x_1-x_3-x_2-x_7}\int_0^{1-x_1-x_3-x_2-x_7-x_6}\frac{C}{D^3}dx_4dx_6dx_7dx_2dx_3dx_1$$

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

C=(x1+x2+x3+x4)*(x4+x5+x6+x7)-x4*x4

cc=x1*m12+x2*m22+x3*m32+x4*m42+x5*m52+x6*m62
+x7*m72

D=-c*cc
.+s*(x1*x2*(x4+x5+x6+x7)+x5*x6*(x1+x2+x3+x4)+x1*x
4*x6+x2*x4*x5)
.+t*x3*x4*x7
.+p12*(x1*x3*(x4+x5+x6+x7)+x3*x4*x5)
.+p22*(x2*x3*(x4+x5+x6+x7)+x3*x4*x6)
.+p32*(x5*x7*(x1+x2+x3+x4)+x1*x4*x7)
.+p42*(x6*x7*(x1+x2+x3+x4)+x2*x4*x7)

テストデータ

m12=1.0d0
m22=1.0d0
m32=1.0d0
m42=1.0d0
m52=1.0d0
m62=1.0d0
m72=1.0d0
p12=1.0d0
p22=1.0d0
p32=1.0d0
p42=1.0d0
s =1.0d0
t =1.0d0

解析近似解=0.1036407209893

aportai

$$I=\int_0^1\int_0^{1-x_1}\int_0^{1-x_1-x_2}\int_0^{1-x_1-x_2-x_3}\int_0^{1-x_1-x_2-x_3-x_4}\int_0^{1-x_1-x_2-x_3-x_4-x_5}\frac{C}{D^3}dx_6dx_5dx_4dx_3dx_2dx_1$$

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

```
C=(x1+x2+x3+x4+x5)*(x1+x2+x3+x6+x7)-(x1+x2+x3)**2

cc=x1*m12+x2*m22+x3*m32+x4*m42+x5*m52+x6*m62
+x7*m72
      D=-c*cc
   .+s*(x1*x2*(x4+x5+x6+x7)+x1*x5*x6+x2*x4*x7-
x3*x4*x6)
   .+t*x3*(-x4*x6+x5*x7)
   .+p12*(x1*x3*(x4+x5+x6+x7)+x3*x4*(x6+x7))
   .+p22*(x2*x3*(x4+x5+x6+x7)+x3*x6*(x4+x5))
   .+p32*(x4*x5*(x1+x2+x3+x6+x7)+x4*x6*(x2+x3)+x1*x5
*x7)
   .+p42*(x6*x7*(x1+x2+x3+x4+x5)+x4*x6*(x1+x3)+x2*x5
*x7)
```

テストデータ

m12=1.0d0
m22=1.0d0
m32=1.0d0
m42=1.0d0
m52=1.0d0
m62=1.0d0
m72=1.0d0
p12=1.0d0
p22=1.0d0
p32=1.0d0
p42=1.0d0
s =1.0d0
t =1.0d0

解析近似解=0.0853513981538

6次元積分実行結果一覧表

演算量　(N=120)

case	GFLOP
laporta g	312064
laporat h	243328
laporta i	232896

実行時間一覧表(秒)

cpu	SR16000		E5-2670		Phi5110P		HD5870
プログラム	32core	64smp	c	fortran	c	fortran	二面
laporta g	2223.744	1418.553	2385.352	2907.167	962.356	943.418	589.471
laporat h	1929.049	1256.823	2096.437	2717.488	925.742	959.321	568.483
laporta i	2212.909	1408.712	2467.079	4071.100	888.431	1063.900	606.803

演算量がこれだけ大きくなると,Phi5110P,HD5870の性能がSR16000を大きく上回ります。また,Phi5110P,HD5870のカタログ性能はほぼ同じですがHD5870の性能がPhi5110Pの倍近く良くなっています。これはPhi5110Pは60CPU,240coreという構成によるものと考えられます。

laporta 計算結果一覧表

演算量　(N=120)

プログラム	GFLOP
LAPORTA D	2514
LAPORTA F	2638
LAPORTA G	312064
LAPORTA H	243328
LAPORTA I	232896

実行時間一覧表(秒)

SR16000

プログラム	32core	64smp
LAPORTA D	8.404	5.426
LAPORTA F	9.841	6.373
LAPORTA G	2223.744	1418.553
LAPORTA H	1929.049	1256.823
LAPORTA I	2212.909	1408.712

E5-2670　HOST　16smp

プログラム	c	fortran
LAPORTA D	27.756	27.621
LAPORTA F	28.275	27.921
LAPORTA G	2385.352	2907.167
LAPORTA H	2096.437	2717.488
LAPORTA I	2467.079	4071.100

Phi　240smp(60core)

プログラム	c	fortran
LAPORTA D	7.474	7.665
LAPORTA F	7.324	7.462
LAPORTA G	962.356	943.418
LAPORTA H	925.742	959.321
LAPORTA I	888.431	1063.900

laporta Phi5110P 実行時間(秒)一覧表

プログラム	1smp /core	2smp /core	3smp /core	4smp /core
laporta d	15.894091	8.930061	8.245382	7.477543
laporta f	15.883854	9.015369	7.781932	7.302511
laporta g	1567.222350	947.766614	956.547483	893.604456
laporta h	1475.760124	890.712335	840.753795	794.503370
laporta i	1682.613524	973.251626	936.788364	908.041320

グラフィックボード性能比較表

実行時間(秒)

プログラム	HD7970	W8000	HD7980
laporta d	7.541418	7.764761	8.173080
laporta f	7.600745	7.821768	8.249741
laporta g	718.902868	738.792693	745.477537
laporta h	684.978227	710.901497	745.486218
laporta i	766.602135	788.137704	834.070693

SR16000/M1　　複数ノード

LAPORTA (D)

MPI数	ノード数	実行時間(秒)
128	4	3.231716
256	4	2.165075
256	8	1.617797
512	8	1.107155

LAPORTA(G)

MPI数	ノード数	実行時間(秒)
128	4	684.465022
256	4	544.483556
256	8	345.817823
512	8	277.552501

LAPORTA (H)

MPI数	ノード数	実行時間(秒)
128	4	541.664086
256	4	387.666776
256	8	270.937051
512	8	198.333612

LAPORTA (I)

MPI数	ノード数	実行時間(秒)
128	4	728.709051
256	4	508.013126
256	8	368.304897
512	8	258.708141

S221　　N=2048

MPI数	ノード数	実行時間(秒)
128	4	467.774509
256	4	412.210191
256	8	233.916246
512	8	206.360301

GPGPU　　LAPORTA

プログラム	実行時間(秒)
D	5.85
G	589.553
H	569.934
I	611.884
S221	2166.834527

BG/Q

LAPORTA(D)

MPI数	ノード数	実行時間(秒)
512	32	12.844877
1024	32	6.633127
2048	32	3.967958

LAPORTA(G)

MPI数	ノード数	実行時間(秒)
512	32	977.217064
1024	32	541.747376
2048	32	439.703755

LAPORTA(H)

MPI数	ノード数	実行時間(秒)
512	32	946.719043
1024	32	514.122564
2048	32	391.466375

LAPORTA(I)

MPI数	ノード数	実行時間(秒)
512	32	954.046258
1024	32	532.563084
2048	32	430.696899

S221　　N=2048

MPI数	ノード数	実行時間(秒)
512	32	2144.31316
1024	32	1161.5669
2048	32	801.675789

理論最大性能(GFLOPs)

BG/Q　6553.6

SR16000　4node　3921.92

8node　7843.84

GPGPU　1088

4.6.5 3loop積分

今回扱った3loop**積分は**

$$\int_0^1 \int_0^{1-x_1} \int_0^{1-x_1-x_2} \int_0^{1-x_1-x_2-x_3} \int_0^{1-x_1-x_2-x_3-x_4} \int_0^{1-x_1-x_2-x_3-x_4-x_5} \int_0^{1-x_1-x_2-x_3-x_4-x_5-x_6} \frac{1}{D^2} d\Omega$$

$$d\Omega = dx_7 dx_6 dx_5 dx_4 dx_3 dx_2 dx_1$$

でDが次の**3ケース**(N0, L01, L02)**の場合です。**

解析解は$20\zeta_5 = 20.7385551028673985...$**。**

3 loop	**実行結果一覧表**	
N = 25		
刻み幅		
N 0	h = 4.734897	/ 24
L 01	h = 4.696708	/ 24
L 02	h = 4.705669	/ 24

SR16000	64smp	4倍精度

プログラム	実行時間(秒)
N0	93.6390
L01	81.0711
L02	67.2363

NO

x8=1.0-x1-x2-x3-x4-x5-x6-x7

x23=x2+x3

x24=x2+x4

x247=x2+x4+x7

x1235=x1+x2+x3+x5

x2346=x2+x3+x4+x6

x5678=x5+x6+x7+x8

D = x24**2*x5**2-x2346*x247*x5**2-x2**2*x2346*x5678
+2*x2*x23*x24*x5678-x1235*x24**2*x5678
-x23**2*x247*x5678+x1235*x2346*x247*x5678
+2*x2*x24*x5*x6-2*x23*x247*x5*x6+x2**2*x6**2
-x1235*x247*x6**2+2*x2*x2346*x5*x7
-2*x23*x24*x5*x7+2*x2*x23*x6*x7
-2*x1235*x24*x6*x7+x23**2*x7**2-x1235*x2346*x7**2

L01

x8=1.0-x1-x2-x3-x4-x5-x6-x7
x123=x1+x2+x3
x167=x1+x6+x7
x348=x3+x8+x4
x2578=x2+x5+x7+x8

D = -x167*x2578*x3**2-x167*x2**2*x348-x1**2*x2578*x348
+x123*x167*x2578*x348-2*x1*x2*x348*x7+x3**2*x7**2
-x123*x348*x7**2-2*x167*x2*x3*x8-2*x1*x3*x7*x8
+x1**2*x8**2-x123*x167*x8**2

L02

x8=1.0-x1-x2-x3-x4-x5-x6-x7
x123=x1+x2+x3
x1456=x1+x4+x5+x6
x678=x6+x7+x8
x247=x2+x4+x7

D = x2**2*x6**2-x123*x247*x6**2-x1456*x2**2*x678
-x1**2*x247*x678+x123*x1456*x247*x678-2*x1*x2*x4*x678
-x123*x4**2*x678-2*x1*x2*x6*x7-2*x123*x4*x6*x7
+x1**2*x7**2-x123*x1456*x7**2

4.6.6 4loop積分

今回扱った4loop積分は

$$\int_0^1 \int_0^{1-x_1} \int_0^{1-x_1-x_2} \int_0^{1-x_1-x_2-x_3} \int_0^{1-x_1-x_2-x_3-x_4} \int_0^{1-x_1-x_2-x_3-x_4-x_5} \int_0^{1-x_1-x_2-x_3-x_4-x_5-x_6} \int_0^{1-x_1-x_2-x_3-x_4-x_5-x_6-x_7} \frac{1}{CD} d\Omega = dx_8 dx_7 dx_6 dx_5 dx_4 dx_3 dx_2 dx_1$$

でC, Dが次の2ケース(M_{44}, M_{45})の場合です。

解析解は

M_{44}は $\frac{441\zeta_7}{8} = 55.5852539156784...$。

M_{45}は $36\zeta_3^{\,2} = 52.017868743610.....$。

演算精度に関しては,解析解と10進10桁以上

合わせるには

M_{44}では倍精度演算で事足ります。

M_{45}では,拡張倍精度,4倍精度演算が必要。

M_{44}**実行結果一覧表**

条件:**分点数**25

刻み幅 $= 4.813324/24$

$t_{max} = 2.406662, \varepsilon^2 = 2^{-49.88652029} \fallingdotseq 9.6 \times 10^{-16}$

E5-2670　16smp　—O1　—OPENMP

言語	精度	実行時間(秒)	備考
FORTRAN	倍精度	206.9346	
C	倍精度	227.7403	
C	拡張倍精度	491.3110	
FORTRAN	4倍精度	22487.9081	
FORTRAN	4倍精度	16751.9997	—O2

Hpi5110P　240SMP　—O1　—OPENMP

言語	精度	実行時間(秒)
FORTRAN	倍精度	1569.4017
C	倍精度	1455.5575
C	拡張倍精度	2334.1927

e5430　8smp　—O1　—OPENMP

言語	精度	実行時間(秒)
C	倍精度	597.7637

M_{45}実行結果一覧表

条件 : 分点数 25

刻み幅 $h = 4.78558154 / 24$

$t_{max} = 2.39279077$, $\varepsilon^2 = 2^{-49.18797875} \fallingdotseq 1.6 \times 10^{-15}$

E5-2670　16smp　−O1 −OPENMP

言語	精度	実行時間(秒)	備考
C	拡張倍精度	688.6699	
FORTRAN	4倍精度	39713.7063	
FORTRAN	4倍精度	30367.7276	−O2
FORTRAN	4倍精度	30745.1106	−O3

SR16000　64smp　−O3 −OMP

言語	精度	実行時間(秒)
FORTRAN	4倍精度	6387.4925

Phi5110P　240 smp　−O1 −OPENMP

言語	精度	実行時間(秒)
C	拡張倍精度	3379.0798

性能的には拡張倍精度演算を使用するのが良い事を示しています。

M_{44}

```
x9 = 1.0 - x1 - x2 - x3 - x4 - x5 - x6 - x7 - x8
     x123 = x1 + x2 + x3
     x167 = x1 + x6 + x7
     x278 = x2 + x7 + x8
     x349 = x3 + x4 + x9
     x589 = x5 + x8 + x9
     C = x167 * x278 * x349 * x589 - x349 * x589 * x7 **2 - x167 * x349 * x8 **2
      - x167 * x278 * x9 **2 + x7 **2 * x9 **2
     D = - (x167 * x278 * x3 **2 * x589) - x167 * x2 **2 * x349 * x589
       - x1 **2 * x278 * x349 * x589
       + x123 * x167 * x278 * x349 * x589 - 2 * x1 * x2 * x349 * x589 * x7
       + x3 **2 * x589 * x7 **2
       - x123 * x349 * x589 * x7 **2 + x167 * x3 **2 * x8 **2
       + x1 **2 * x349 * x8 **2
       - x123 * x167 * x349 * x8 **2 - 2 * x167 * x2 * x3 * x8 * x9
       - 2 * x1 * x3 * x7 * x8 * x9
       + x167 * x2 **2 * x9 **2 + x1 **2 * x278 * x9 **2
       - x123 * x167 * x278 * x9 **2
       + 2 * x1 * x2 * x7 * x9 **2 + x123 * x7 **2 * x9 **2
```

M_{45}

```
x9 = 1.0 - x1 - x2 - x3 - x4 - x5 - x6 - x7 - x8
    x123 = x1 + x2 + x3
    x1567 = x1 + x5 + x6 + x7
    x279 = x2 + x7 + x9
    x349 = x3 + x4 + x9
    x589 = x5 + x8 + x9
    c = -x279 * x349 * x5 * *2 + x1567 * x279 * x349 * x589 - x349 * x589 * x7 * *2
     - 2 * x349 * x5 * x7 * x9 - x1567 * x279 * x9 * *2 - x1567 * x349 * x9 * *2
     + x5 * *2 * x9 * *2 - x1567 * x589 * x9 * *2 + 2 * x5 * x7 * x9 * *2
     + x7 * *2 * x9 * *2 + 2 * x1567 * x9 * *3
    d = x279 * x3 * *2 * x5 * *2 + x2 * *2 * x349 * x5 * *2 - x123 * x279 * x349 * x5 * *2
       - x1567 * x279 * x3 * *2 * x589 - x1567 * x2 * *2 * x349 * x589
       - x1 * *2 * x279 * x349 * x589 + x123 * x1567 * x279 * x349 * x589
       - 2 * x1 * x2 * x349 * x589 * x7 + x3 * *2 * x589 * x7 * *2
       - x123 * x349 * x589 * x7 * *2 + 2 * x1 * x279 * x3 * x5 * x9
    - 2 * x1 * x2 * x349 * x5 * x9 + 2 * x2 * x3 * x5 * *2 * x9 - 2 * x1567 *
    x2 * x3 * x589 * x9
    + 2 * x2 * x3 * x5 * x7 * x9 + 2 * x3 * *2 * x5 * x7 * x9 - 2 * x123 * x349 *
    x5 * x7 * x9
    - 2 * x1 * x3 * x589 * x7 * x9 + x1567 * x2 * *2 * x9 * *2 + x1 * *2 * x279 * x9 * *2
    - x123 * x1567 * x279 * x9 * *2 + 2 * x1567 * x2 * x3 * x9 * *2
    + x1567 * x3 * *2 * x9 * *2 + x1 * *2 * x349 * x9 * *2 - x123 * x1567 * x349 * x9 * *2
    + 2 * x1 * x2 * x5 * x9 * *2 - 2 * x1 * x3 * x5 * x9 * *2 + x123 * x5 * *2 * x9 * *2
    + x1 * *2 * x589 * x9 * *2 - x123 * x1567 * x589 * x9 * *2 + 2 * x1 * x2 * x7 * x9 * *2
    + 2 * x1 * x3 * x7 * x9 * *2 + 2 * x123 * x5 * x7 * x9 * *2 + x123 * x7 * *2 * x9 * *2
    - 2 * x1 * *2 * x9 * *3 + 2 * x123 * x1567 * x9 * *3
```

多倍長計算手法まとめ

平成23年度-平成27年度

多倍長計算手法まとめ

1.はじめに

倍精度演算で正しい結果が得られない場合の
原因としては以下の2つの場合が考えられます。

(1) 近接する2変数の減算による桁落ち
(2) 演算中に表現できる数値範囲を超える場合

対策としては
(1) 倍精度浮動小数点数を複数個つなげた演算を行う
(2) ieee754-2008形式の4倍精度変数を拡張する。
があり,(1)をDD形式(浮動型),(2)をieee形式(整数型)と称します。場合によっては,(2)の形式を複数個つなげる方式(DQ形式)を使用します。

どちらの方式も一長一短がありすべてのケースを網羅する方式は現在はないと言えます。

また並列化により発生する問題として,指数部の差が大きい場合の加算での演算順序や部分和の採り方により逐次実行と結果が大きくと異なる場合があります。それもかならずしも並列化時の精度が悪いわけでもないと言う厄介な問題もあります。
簡単な問題でその例を示します。

```
c
      implicit real*4 (a-h,o-z)
      s=0.0
      a=1.0
      do i=1,100000000
      s=s+a
      end do
      write (6,1)  s
    1 format (1h ,2x,f20.5)
      stop
      end
```

正しい結果は100000000ですが次ページ以降に示すように最適化オプションや並列化で様子が異なります。

SR16000/xm1 f90　並列化をしない場合

```
% f90 a.f -O3 -o a
f90: compile start : a.f

*OFORT90 V03-02-/B entered.
*program name = MAIN
*end of compilation : MAIN
% ./a &
%     16777216.00000


% f90 a.f -Os -noloopexpand -
noparallel -o a
f90: compile start : a.f

*OFORT90 V03-02-/B entered.
*program name = MAIN
*end of compilation : MAIN
% ./a
    16777216.00000
```

```
% f90 a.f -Os -noparallel -o a
f90: compile start : a.f

*OFORT90 V03-02-/B entered.
*program name = MAIN
*end of compilation : MAIN
% ./a
       100000000.00000
```

SR16000/xm1 f90　並列化をした場合

```
% f90 a.f -Os -o a
f90: compile start : a.f

*OFORT90 V03-02-/B entered.
*program name = MAIN
*end of compilation : MAIN
*end of compilation :
_parallel_func_1_MAIN
% ./a
       100000000.00000
%
```

次にスレッド数指定で異なる例を以下の例で示します。先のプログラムの100000000を1000000000に変更しただけのものです。

```
c
      implicit real*4 (a-h,o-z)
      s=0.0d0
      a=1.0d0
C$OMP PARALLEL DO
C$OMP&  reduction (+:s)
      do i=1,loop
      s=s+a
      end do
C$OMP END PARALLEL DO
      write (6,1)  s
    1 format (1h ,2x,f20.5)
      stop
      end
```

E5－2670

16smp

$268435456.00000 = 2^{28} = 2^{24} \times 16 < 10^{9}$

より結果は $2^{28} = 2^{24} \times 16$**となる。**

100smp

$1000000000.00000 = 10^{9} < 2^{24} \times 100$

より正しい結果となる。

Phi5110P

100smp　1000000000.00000

240smp　1000000000.00000

ここで 2^{24} **は単精度加算演算で正確に計算できる個数となっています。**

2. 多倍長精度演算

演算方式には,浮動小数点演算方式と整数演算方式がある.

2.1 浮動小数点演算方式

多倍長精度変数を複数個の倍精度変数の和で表す。
表現できる数値範囲は,ieee754の倍精度と同じ.

a:$2n$倍精度変数

$a_1, a_2, ..., a_n$：倍精度変数

$a = a_1 + a_2 + ... + a_n$

有効ビット数＝仮数部のビット数＋1

$2n$倍精度$(52+1)\times n$　　ビット

$n=2$　　106ビット

$n=3$　　159ビット

$n=4$　　212ビット

乗加算命令がある場合とない場合の処理に分かれる。
コンパイラの最適化方式に性能が大きく依存する。

次に示す基本演算例よりわかります。

基本の演算

倍精度変数の 加減算,乗除算の結果を2つの 倍精度変数の和で表す。

a,b,c,d**倍精度変数**; $a \pm b, a \times b, a \div b = c + d$

加算 : $c = a + b$

$t = c - a$

$d = (a - (c - t)) + (b - t)$

減算 : $c = a - b$

$t = c - a$

$d = (a - (c - t)) - (b - t)$

加算,減算ともに最適化オプションで括弧をはずした演算順序に ならない 様に注意が必要

乗算(乗加算命令あり.)

$c = a * b$

$d = a * b - c$ **(最適化オプションにより**$d = 0$**とならないように 注意が必 要)**

乗算(乗加算命令なし)

$r = 134217729.0d0 \quad (= 2^{27} + 1)$

$c = a * b$

$t1 = a * r$

$a1 = t1 - (t1 - a)$

$a2 = a - a1$

$t2 = b * r$

$b1 = t2 - (t2 - b)$

$b2 = b - b1$

$d = ((a1 * b1 - c) + a1 * b2 + a2 * b1) + a2 * b2$

演算量	加減算	n^2に比例
	乗算	n^3に比例

$n \geq 3$では

n個の変数の絶対値が大きい順にならべるソート処理が必要.

2.2 整数演算方式

多倍長精度変数はieee754-2008の4倍精度の仮数部を変えたものであらわす。
表現できる数値範囲はieee754-2008の4倍精度と同じ.

a : p倍精度($p \geq 3$)変数

符号部　1ビット

指数部　15ビット

仮数部　$32 \times p - 16$ビット

有効ビット数

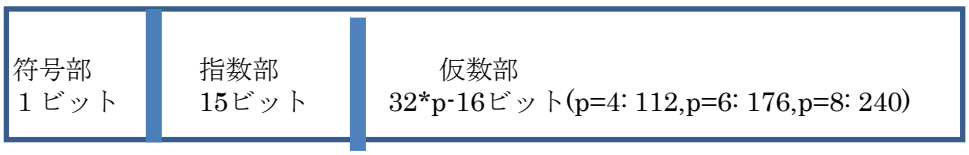
$p = 4$　113ビット

$p = 6$　177ビット

$p = 8$　241ビット

p 倍精度

演算量	加減算	pに比例
	乗算	p^2に比例



2.3　ルーチン作成前に決定すべき事項

ルーチン作成にあたって以下の項目を決めて置く必要があります。

（1）最終ビット処理
　　現在まで一般的に使用されていた
　　丸め処理を適用
（2）アンダーフロー,オーバーフロー処理
　　浮動小数点演算方式では既存の倍精度
　　演算と同じとし,整数演算方式では
　　ともに結果を0とし,オーバーフロー
　　は使用者責任とした。
（3）ゼロ割り処理
　　浮動小数点演算方式では既存の
　　倍精度演算と同じとし,整数演算方式
　　ではすべて結果を0としこれも使用者
　　責任とした。
　　0/0=0,a/0=0.

2.4　ルーチン作成の要点

最初に加減算,乗算ルーチンを作成し平方根は既存の倍精度演算を使用して多倍長演算の初期値として反復法により求める。除算に関しては,浮動小数点演算方式では既存の倍精度除算を複数回使用する。6倍精度以上（使用する倍精度変数が3以上の場合,絶対値が大きい順になる様にする注意が必要。整数演算方式では逆数の初期値を既存の4倍精度除算を使用して反復法により逆数を求め,被除数との積をとり求める。

数学関数等の作成は上記の5演算の組み合わせで行う。具体的に立方根,対数関数,指数関数,正弦関数計算でその注意事項を記述する。

引数の取り得る範囲(ここでは表現可能な数値の絶対値
最大値をMとする)は立方根,指数関数,正弦関数は $|x| \leq M$.
対数関数は $0 < x \leq M$。ここで関数値yのとる範囲は,
立方根,対数関数は $|y| \leq M$,指数関数は $0 < y \leq M$。
正弦関数は $|y| \leq 1$。ここで,指数関数は $x \geq 0$ で $y > e^x$
より $0 \leq x \leq \log(M), e^{-x} = \frac{1}{e^x}$ より $|x| \leq \log(M)$ となる.
立方根に関しては $x = y \times z$ (yの立方根は容易に計算出来る
数値,zをある標準区間)とすると,$\sqrt[3]{x} = \sqrt[3]{y} \times \sqrt[3]{z}$ より,$x = y \times z$ の
分解によって誤差は発生しにくい。立方根に関してはこの関数
が準備されていないシステムもあり,$\sqrt[3]{x} = e^{\frac{1}{3}x\log(x)}$ (べき乗)で計算
される事があるが $x \leq 0$ ではエラーとなるので注意が必要である。
判れば簡単な事ではあるが,多くの人が同じ間違いを繰り返した
問題であるので注意が必要である。
指数関数及び正弦関数は
テーラー展開式があるが,引数範囲すべてで計算すると
性能,精度の問題が発生する。また対数関数では引数範囲の
テーラー展開はなく,$x = y + z$ の様に分解してある近似区間でz
で計算してxを求めるが,$e^x = e^y \times e^z$ なので指数関数は誤差が
発生しにくい。対数関数と正弦関数を比較すると,結果の範囲
が $|y| \leq M$,と $|y| \leq 1$ から,正弦関数の計算の方が誤差が出易い
事がわかる。正弦関数の計算方法の一例をあげる。
$I = [|x| / \frac{\pi}{2}], t = |x| - I \times \frac{\pi}{2}$ とし $0 \leq t < \frac{\pi}{2}, 0 < \frac{\pi}{2} - t \leq \frac{\pi}{2}$ より
$\sin(x) = \pm\sin(t), \pm\sin(\frac{\pi}{2} - t)$ で計算する。
ここで問題なのは,$|x|$ が大きい場合,xとtの持つ精度が異なる
ことである。(tを求める際の桁落ちのため).また,$|x|$ が小さくても
xが $n\pi$ に近い場合,tは桁落ちが発生するがこの場合は $\sin(x) \fallingdotseq 0$
のため,相対誤算は大きいが絶対誤差は小さくて済む。精度上の
問題が発生した場合には,作成者により精度に関する思想、扱い
が異なるという意識を持つことが必要となる。

2.5 注意点詳細

(1)浮動小数点演算方式

(ア)小さな数は精度が保持できない場合がある。

$1/10^{-305}$などでは6倍精度,8倍精度演算では

精度分の結果を得る事が出来ない。

(イ)性能やソース作成の手間を考慮すると,

12倍精度演算までにする必要がある。

(ウ)数値表現の制限にかかる場合は,拡張倍精度,

ieee754 – 2008の4倍精度の変数をつなげた

演算を行う必要がありこの場合はつなげる個数

は2つまで。

(2)整数演算方式

(ア)プログラム作成でもっとも間違えやすいのは,

近接する2つの数の減算での桁落ちの数を求める部分。

(イ)4倍精度演算で一付近の数xに対し1 – xの桁落ちが発生

する場合には,1変数の8倍精度演算より4倍精度変数を

2つつなげた8倍精度演算を使用した方が良い。

拡張倍精度変数を2つつなげた演算で行えれば、性能

的にはより有効な方法となる。

(ウ)(イ)と同じ様なケースが4倍超精度演算で発生する場合

1変数のさらに高精度な演算ではなく,4倍超精度変数を2

つつなげた高精度な演算を使用した方が良い。

2.6 整数演算方式の並列化に関して

整数演算方式では,浮動小数点数a,bを正整数ia,ibに変換して,iaとibの加減算,乗算,除算を行なう.加減算ではia,ibに変換する(ia ≥ ib)際,a,bの値によりibのシフト数が定まる。
一組の数に対する操作 では問題ないが、simdの様に1core内で複数組の数を並列 に操作する場合には,1core内で各組毎に異なるシフト処理が出来る様にする必要がある。
複数coreで複数組の数を並列に 操作する場合は各core毎に異なるシフト処理ができるので問題はない。これはsimdと言ってもSR16000,BG/QとHD5870の様なGPUでは,適用範囲が異なる事を意味する。乗算,除算に関しては複数組 の数で異なる処理はない。演算後の 処理では、加算,乗算,除算は桁上げ(加算,乗算),桁下げ(除算)があるかないかの2ケースしかないので,1core内でのsimdが適用できる様にするには,マスク処理が出来る様にする必要がある。減算の場 合,桁下げの数を求めるのに,左から検索して最初に1が現れるビット位置をもとめる処理が必要になる。1core内でのsimdが適用できる様にするには,この処理が出来る様にする必要がある。
多倍長整数演算の場合 ,nビット(n >> 1)を複数ビット毎に分けて演算するが,加減算,乗算は市販本などでよく知られている方法で容易に演算出来るが,除算では特別なアルゴリズムが必要となる。また除 算では逆数(1 ≤ a < 2)の初期近似を既存の浮 動小数点除算で求め,$x_{n+1} = x_n(2 - ax_n)$で$1/a$を求めて,$b/a = b \times 1/a$で除算を行う方法がある。$2 - ax_n$ ≒ 1なので$2 - ax_n$の桁下がりはあるかないかの2ケースとなり,1core内でのsimd処理には,他の演算と同じ様にマスク処理 があれば良い。これまでの実行では除算に関しては,3倍精度から8倍精度演算ではビット数が固定なら特別なアルゴリズムの使用した除算,可変なら任意の整数除 算を使用した除算,8倍精度超精度演算では、既存の除算命令と反 復法で逆数を求めて除算を行うのが良いという結果となっている。

これを以下のRump’s例題を使用して検証しました。

Rump's**例題**

$a = 77617.0 \qquad b = 33096.0$

$$f = 333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b}$$

理論解 $= -\frac{54767}{66192} = -0.827396059946$

$$[333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2)] + 5.5b^8$$

の演算で121**ビットの桁落ちが発生。**
桁落ちが影響を受けると0,**受けないと** -2**となる。**
4**倍精度演算では有効ビット数が**113**のため**
桁落ちの影響を受けるため1.172603940053**となり**,
有効ビット数が121**より大きい演算**
(5**倍超精度演算)では桁落ちの影響を**
受けないので -0.827396059946**となる。**

Core内simdは適用できないのでcore間smpで実行している。

多倍長rump's例題一覧表

1,000,000回実行

実行時間一覧表(秒)

演算精度	E5-2670 16smp	Phi5110P 4smp/core 240smp
68倍精度	11.9698	46.3840
128倍精度	42.3808	156.0239
188倍精度	89.0890	331.2860
248倍精度	152.0543	585.7436
308倍精度	248.2340	966.5418
368倍精度	355.4690	1378.1645
428倍精度	476.9385	1859.8102
488倍精度	617.0996	2410.4360
548倍精度	833.8722	3244.1159

Core間のスレッド並列の効果が出ている。

多倍長rump's例題一覧表
1,000,000**回実行**

実行時間一覧表(秒)

E5-2670

演算精度	16smp	8smp	4smp	2smp	1smp
68倍精度	11.9698	23.8582	47.8585	95.5189	191.5384
128倍精度	42.3808	84.7253	169.0772	338.4619	677.3167
188倍精度	89.0890	177.8675	355.6048	711.6836	1425.8922
248倍精度	152.0543	303.7099	607.2627	1214.3922	2438.3971

Phi5110P 4smp/core

演算精度	240smp	180smp	120smp	60smp
68倍精度	46.3840	48.7955	50.8138	69.1542
128倍精度	156.0239	178.3118	181.1152	244.4755
188倍精度	331.2860	378.7431	384.5338	525.7136
248倍精度	585.7436	669.6312	694.6440	938.0996

演算精度	4smp	1smp
68倍精度	868.7135	3463.0626

3.実アプリケーション

3.1 ヒルベルト行列Hによる連立一次方程式Hx=bの求解

3.1.1 ヒルベルト行列の条件数

ヒルベルト行列 $H_{ij} = \frac{1}{i+j-1}\ (1 \le i, j \le n)$

逆行列 $H_{ji}^{-1} = (-1)^{i+j} \frac{(n+i-1)!(n+j-1)!}{(i+j-1)[(i-1)!(j-1)!]^2 (n-i)!(n-j)!}$

H_{ij}**のノルムは** $1 + \frac{1}{2} + ... + \frac{1}{n}$ **nが大とすると**,

$\ln(n) + \gamma \quad \gamma = 0.57721566490153286060$ (**オイラー数**)

逆行列 H_{ji}^{-1} **のノルムを** $|H_{ji}^{-1}|$ **の最大値とする。**

またnは大として、**スターリングの公式** $n! \sim \sqrt{2\pi} e^{-n} n^{n+\frac{1}{2}}$ **を使用する。**

逆行列の絶対値Pは**変形して**,

$$P = \frac{(n+i)!(n+j)!i^2 j^2}{(n+i)(n+j)(i+j-1)[i!\,j!]^2 (n-i)!(n-j)!}$$

$i = an, j = bn$**とすると**,

$$P = \frac{a^2 b^2 n^2 ((1+a)n)!((1+b)n)!}{((a+b)n-1)(1+a)(1+b)[(an)!(bn)!]^2 ((1-a)n)!((1-b)n)!}$$

$$= \frac{ab\sqrt{\frac{1+a}{1-a}}\sqrt{\frac{1+b}{1-b}}}{(1+a)(1+b)((a+b)n-1)4\pi^2} (\frac{1+a}{1-a})^{(1-a)n} (\frac{1+b}{1-b})^{(1-b)n} (\frac{1+a}{a})^{2an} (\frac{1+b}{b})^{2bn}$$

Pは**a**と**b**に**対して対称より**, $a = b$**とすると、**

$$P = \frac{a^2}{(1-a^2)(2an-1)4\pi^2} (\frac{1+a}{1-a})^{2(1-a)n} (\frac{1+a}{a})^{4an}$$

$f(a) = \ln[(\frac{1+a}{1-a})^{(1-a)} (\frac{1+a}{a})^{2a}]$**とすると** $f'(a) = \ln(\frac{1-a^2}{a^2})$

$0.5 \le a \le 0.75$**では**$f(a)$**は**$a = \sqrt{0.5} = 0.7071$**で最大となる**.

$i = an, j = bn$**が整数となる事から**$, a = 0.7$**とする**.

すると$P_{max} = \dfrac{0.0243}{1.4n-1}(\dfrac{17}{7})^{2.8n}(\dfrac{17}{3})^{0.6n} = \dfrac{0.0243}{1.4n-1}10^{1.530979n}$

より詳細にすると以下の様になる。

$$a = 0.7 \qquad P = \frac{0.02433695097}{1.4n-1}(\frac{17}{3})^{0.6n}(\frac{17}{7})^{2.8n}$$

$$= 10^{1.530979068n-1.613733833-\log_{10}(1.4n-1)} \quad (1)$$

$$a = \sqrt{0.5} = \frac{\sqrt{2}}{2} \qquad P = \frac{1}{(\sqrt{2}n-1)4\pi^2}(\frac{2+\sqrt{2}}{2-\sqrt{2}})^{(2-\sqrt{2})n}(\sqrt{2}+1)^{2\sqrt{2}n}$$

$$= 10^{1.531102741n-1.596359739-\log_{10}(\sqrt{2}n-1)} \quad (2)$$

$$(2)/(1) = 10^{0.000123673n+0.017374096+\log_{10}(1.4n-1)-\log_{10}(\sqrt{2}n-1)}$$

10^m**の**m

	(1)	(2)	(2)/(1)	**条件数比率**
$n = 200$	302.1364756	302.1741817	0.0377	1.091
$n = 400$	608.0304816	608.0929301	0.0624	1.155

近似的には条件数 $= 10^{1.5N}$ **となる**.

条件数は$Ax = b$を$(A + \Delta A)(x + \Delta x) = (b + \Delta b)$
で計算する場合のΔxの大きさに関する目安となる
ものである。ヒルベルト行列H(ここではAとする)の係数
を8倍精度演算で求めるとします.ここで解$x = (1,1,.....,1)^T$
となる様にbを8倍精度演算で計算します.すなわち
$\Delta A, \Delta b \neq 0$の場合,8倍精度演算で計算すると,

DIMENSION SIZE = 40
int 8bai seido
gosa = 5.4539714738897539969293481897244 12E - 0016
DIMENSION SIZE = 60
int 8bai seido
gosa = 86.17035103143253639375884979092 26

となる.
ここでAの係数を4倍精度演算でもとめると,$\Delta A, \Delta b = 0$となり

DIMENSION SIZE = 200
int 8bai seido
gosa = 3.3059183771269235935702822166686 63E - 0036

Aの係数を倍精度演算で求め,他の演算を
4倍精度で行った場合でも
DIMENSION SIZE = 200
GOSAMAX = 7.5680989840885551439816886486970 12E - 0017
となる。

この事は,入力データは低精度(もともとデータで10進30桁の精度
がある事は稀である.)で演算のみを高精度で行い,計算結果を
低精度でも出力するという場合に適用できる。
多倍長精度演算のテストでは,$\Delta A, \Delta b \neq 0$の場合で実施。

3.1.2 ヒルベルト行列実測結果
3.1.2.1 64倍精度演算までの結果

ヒルベルト行列最大誤差一覧表

次元数	倍精度	4倍精度	8倍精度
20	51.616	1.44E−07	2.13E−47
40	182.107	62.034	8.72E−16
60	1971.569	62.804	251.132
80	642.074	300.734	252.653
100	201.355	378.428	242.672

次元数	16倍精度	32倍精度
20	1.47E−99	1.44E−276
40	4.09E−86	4.30E−243
60	2.66E−60	1.36E−210
80	3.65E−31	1.08E−178
100	2.58E+00	1.41E−150

ヒルベルト行列の精度

N	演算精度	最大誤差	条件数 (ビット数表示)
170	32倍精度	4.11E−48	853
180	32倍精度	2.00E−32	904
190	32倍精度	7.11E−17	955
200	32倍精度	1.27E−02	1006
210	32倍精度	1.37E+02	1057
370	64倍精度	9.63E−35	1870
380	64倍精度	7.98E−30	1920
390	64倍精度	7.61E−15	1971
400	64倍精度	7.10E+01	2022

3.1.2.2　128-248倍精度演算結果

ヒルベルト行列の精度		
N	最大誤差	精度
400	1.014E-621	128倍精度
500	1.681E-467	128倍精度
600	1.321E-314	128倍精度
700	1.349E-161	128倍精度
800	7.603E-009	128倍精度
1000	1.013E-280	188倍精度
1100	4.211E-127	188倍精度
1400	1.013E-245	248倍精度

最大誤差のビット数 = 有効ビット数 – 条件数のビット数となっています。

248倍精度演算では$N = 1400$が使用できるメモリ容量の上限となっています。

3.1.2.3 308-488倍精度演算での結果

N=100でのヒルベルト行列計算

条件数0.127E+152　　ビット数501

精度	有効ビット数	誤差		
		実測	理論値	
308	9841	4.73E-2814	2.80E-2812	
368	11761	3.38E-3392	2.52E-3390	
428	13681	1.61E-3971	2.66E-3968	
488	15601	5.36E-4198	2.80E-4546	(注)
488	15601	1.00E-4547	2.80E-4546	

(注)逆数計算で4倍精度の初期値から7回反復した場合の値

308,368,428倍精度は7回反復,488倍精度は8回反復が必要

3.1.2.4　548倍精度演算での結果

誤差は解$x = (1,1,....,1)^T$と比較していますので、小さなNに対しては差が0と正しく計算しているかの検証とは言えません。そこでNを変えながら正しい事を検証しました。

x5570

最大誤差	実行時間（秒）	条件数（ビット数）	有効ビット−条件数
0.000D−00	182.566	501	0.000D−0000
0.000D−00	1304.570	1010	0.000D−0000
1.647D−48	4368.843	1518	4.140D−4818
1.656D−46	10544.205	2022	2.168D−4666
4.445D−45	21630.467	2530	1.817D−4513

誤差が出るサイズ

条件数1138ビットとなるN

N	条件数（ビット数）
223	1127
224	1132
225	1137
226	1142
227	1147
228	1152

3.2 対称疎行列Aによる連立一次方程式 Ax=bの求解

扱った問題

ポアソン方程式

$$\Delta u = -f \quad (\Omega), u = 0 \quad (\partial\Omega)$$

$$\Omega = [0,1] \times [0,1] \times [0,1]$$

$$N = 200 \times 200 \times 200$$

収束判定値：**共役残差** 10^{-12}

$$u = \sin(\pi x)\sin(\pi y)\sin(\pi z)$$

$$f = 3 \times \pi^2 \times \sin(\pi x)\sin(\pi y)\sin(\pi z)$$

初期値は $x(i) = 1.0$

$cgs, cgsilu, cgsmil$**の非対称疎行列用反復法では収束しない。**

この様な場合，初期値は

$x(i) : (0,1)$**の一様乱数値としている。**

倍精度演算実行での反復回数

		反復回数	
	解法	反復回数	備考
	sor	4293	陰解法
	sor2	141799	陽解法
	sor3	4275	陰解法+odd-even法
	adi	34195	陽解法
	cg	690	
	bcg	704	
	bicgs	495	
	cgs	640	初期値x(i)=1.0では収束しない。
	cgsm	288	cgsiluの並列版
	前処理付き		
	scg	693	前処理付きの最も簡単なケース
	cgsmil	241	初期値x(i)=1.0では収束しない。
	cgsilu	272	初期値x(i)=1.0では収束しない。
	cgsmm	306	cgsmilの並列版
	gpbicg	142	

並列実行にすると，初期値$x(i)=1.0$でも収束するように改善される場合がある。
反復回数の削減には，
初期値の変更，並列化などがある事になる。

3.3 非対称疎行列Aによる連立一次方程式 Ax=bの求解

問題は3次元ポアソン方程式

$$\Delta u + R\frac{\partial u}{\partial x} = -f$$

領域$[0,1]\times[0,1]\times[0,1]$

解析解$u(x,y,z) = e^{xyz}\times\sin(\pi x)\times\sin(\pi y)\times\sin(\pi z)$

非対称問題。

解法

bcg(biconjugate gradient)**法**

$R = 100, nx = ny = nz = 65$

$R = 100, nx = ny = nz = 129$

cgs(conjugate gradient square)**法**

$R = 100, nx = ny = nz = 65$

$R = 100, nx = ny = nz = 113$

$R = 100, nx = ny = nz = 129$

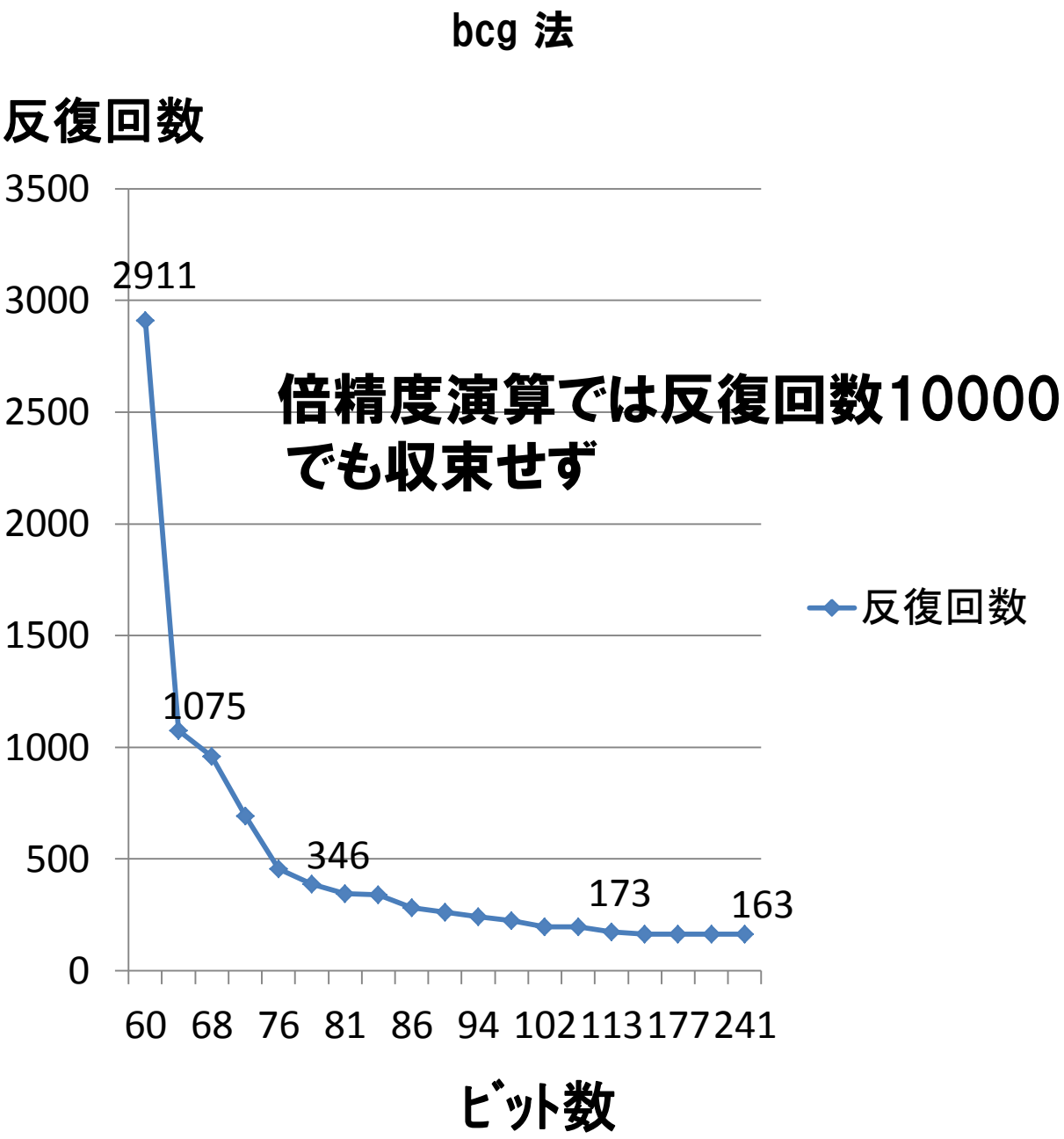
収束判定値は共役残差10^{-12}

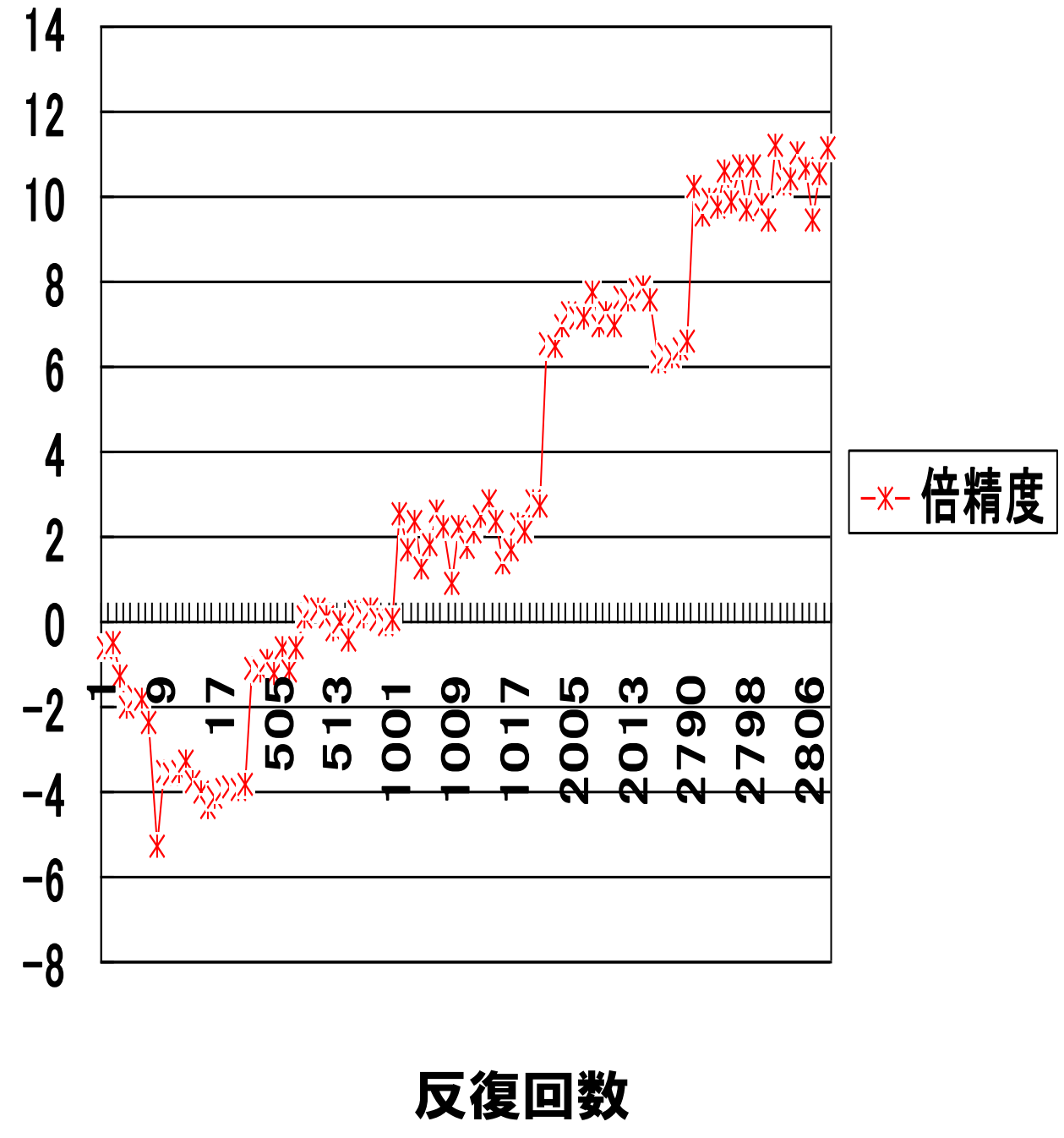
初期値$u(x,y,z) = 0$

サイズは精度改善により拡大しています。

3.3.1 nx=ny=nz=65 の場合

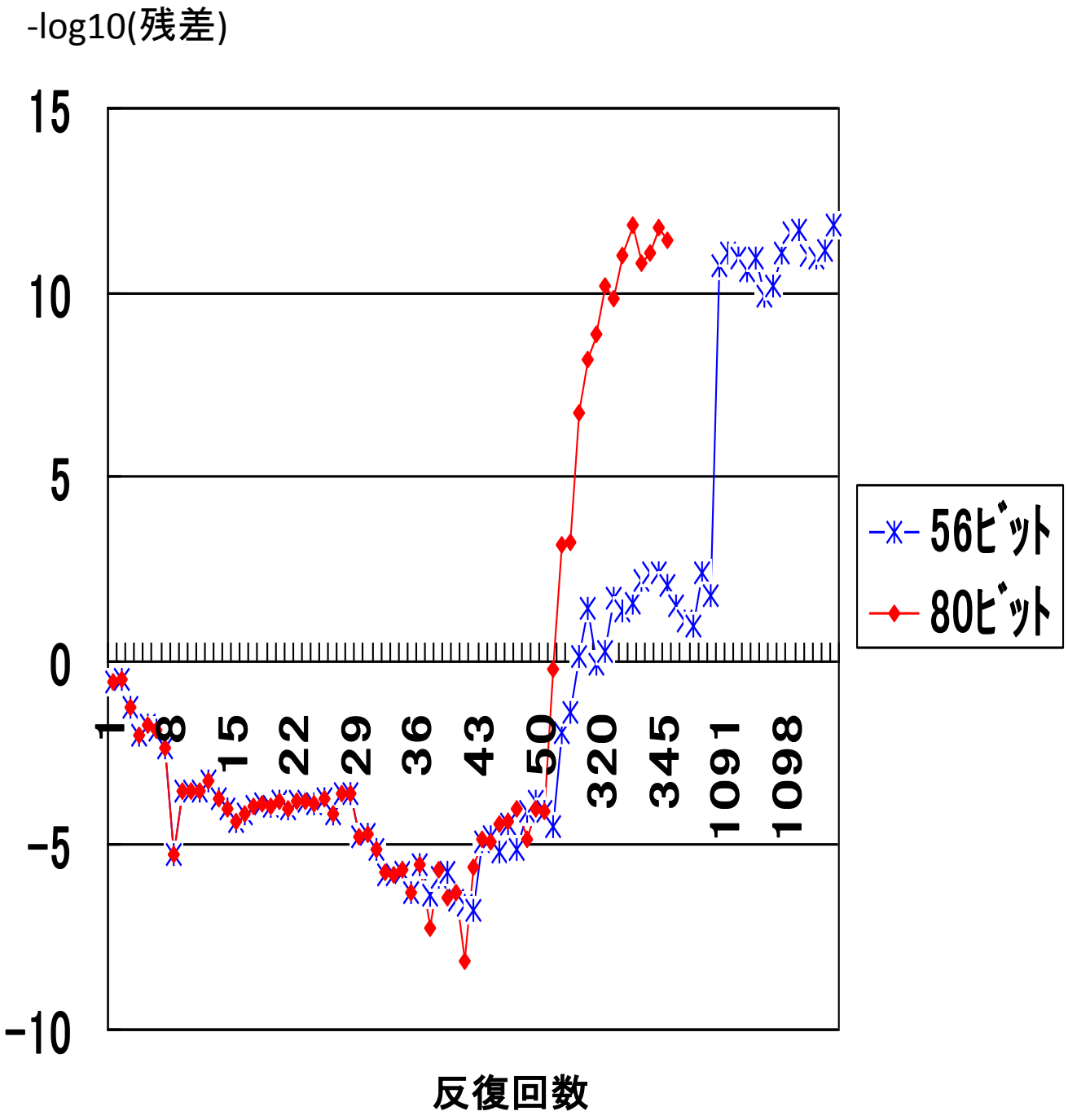
3.3.1.1 BCG法の収束状況

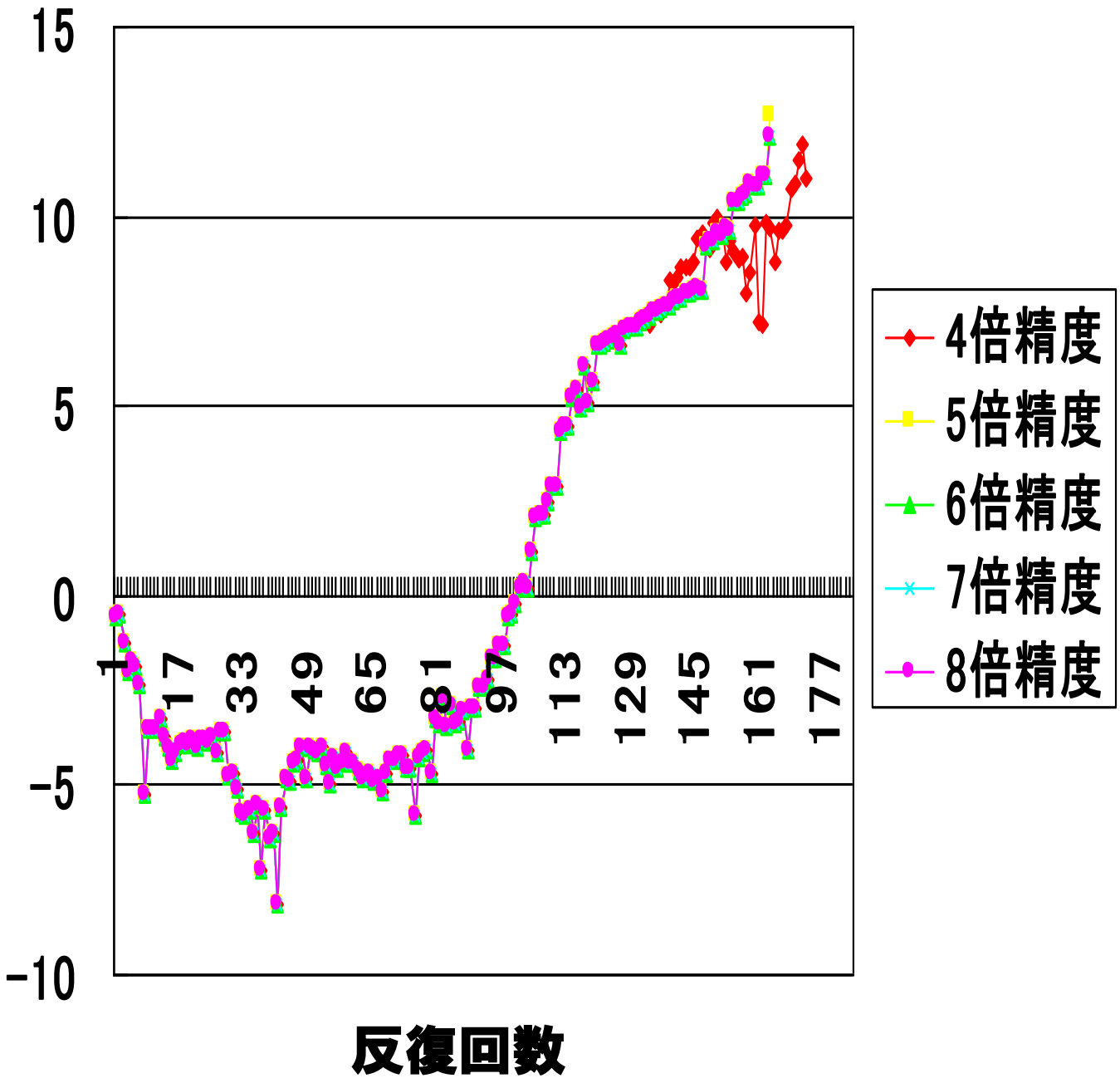



-log10(残差)
14
12
10
8
6
4
2
0
-2
-4
-6
-8
1
9
17
505
513
1001
1009
1017
2005
2013
2790
2798
2806
倍精度
反復回数

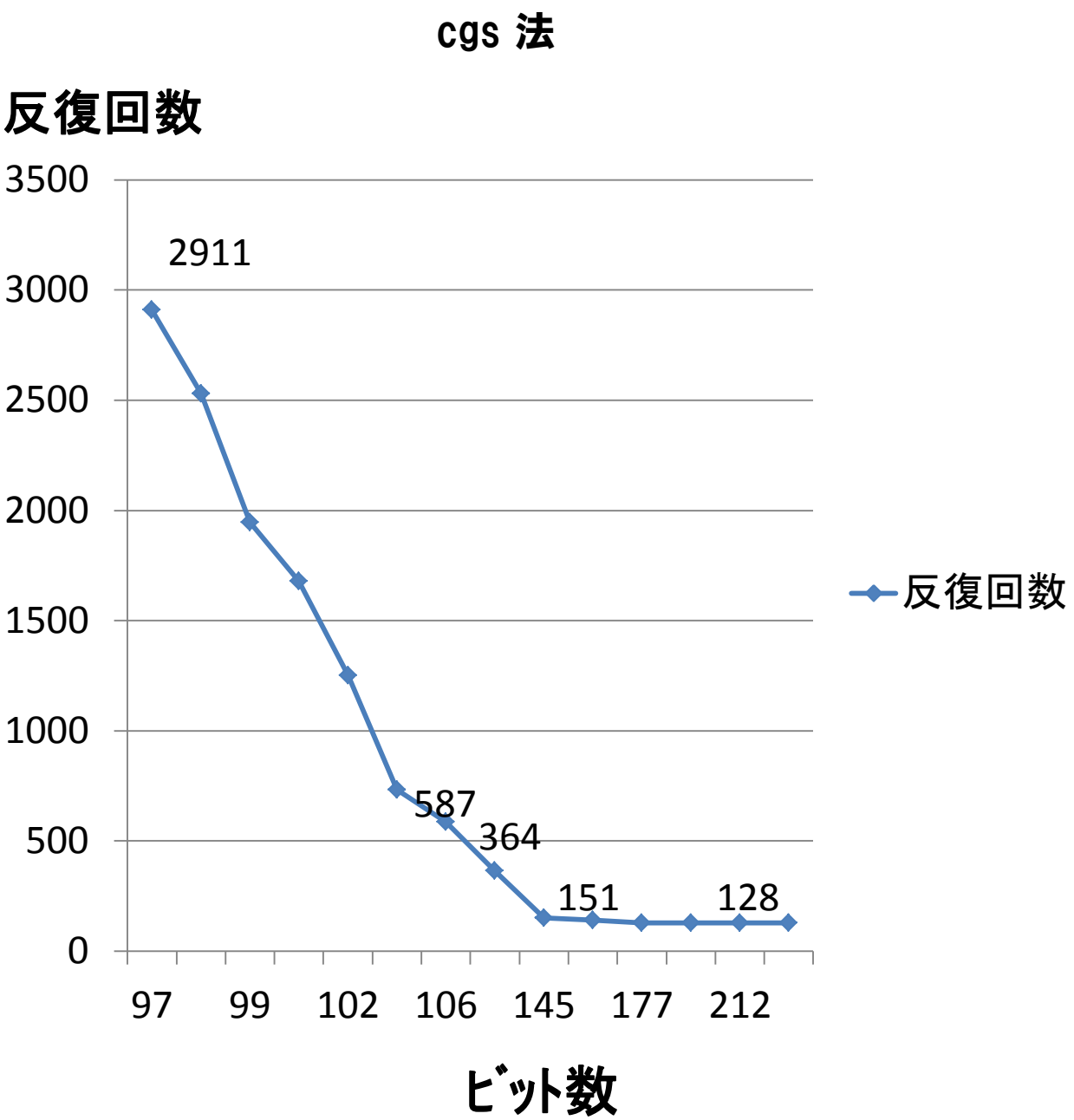
BCG法の収束状況

nx=ny=nz=65 収束判定値=1.0q-12

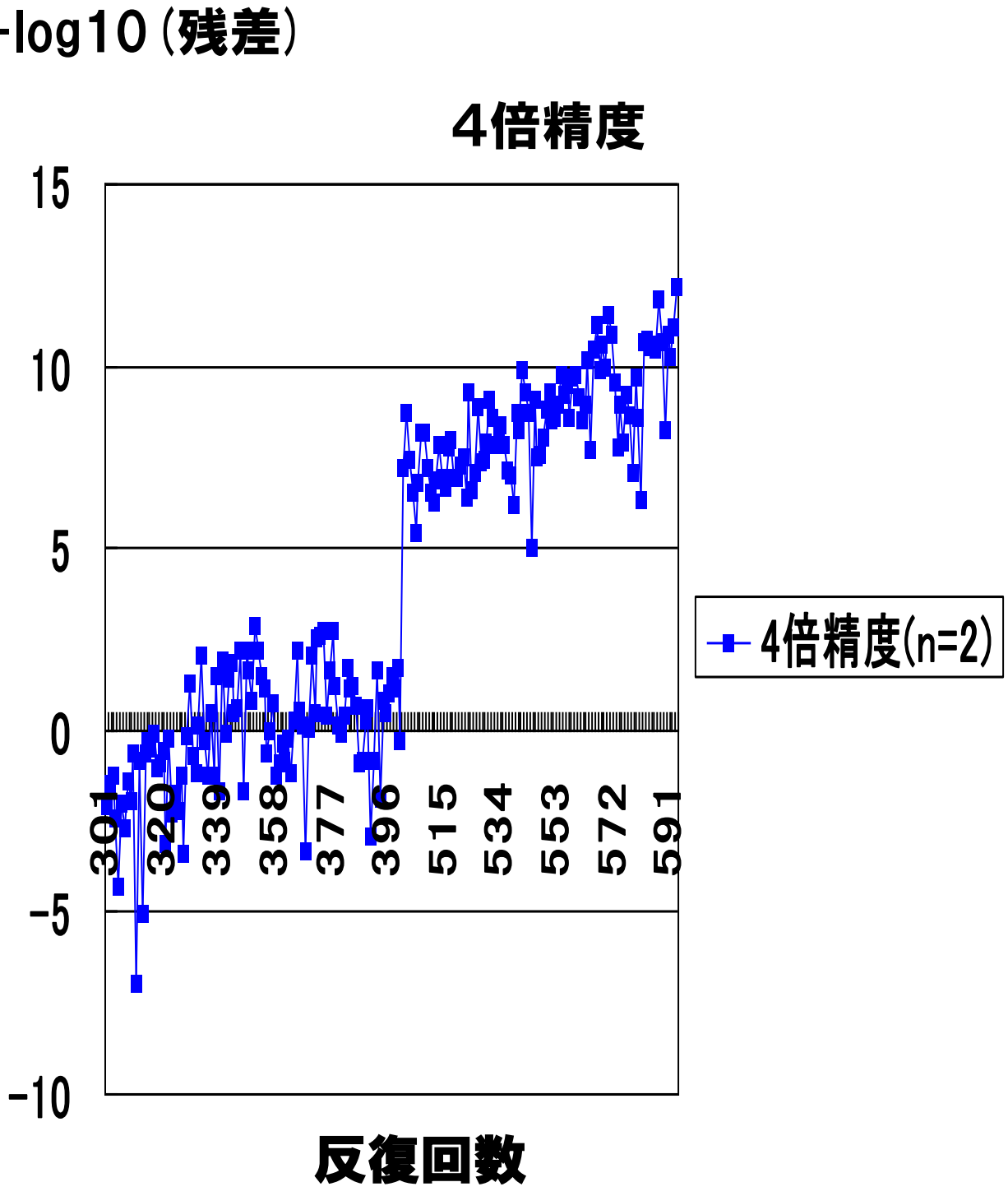


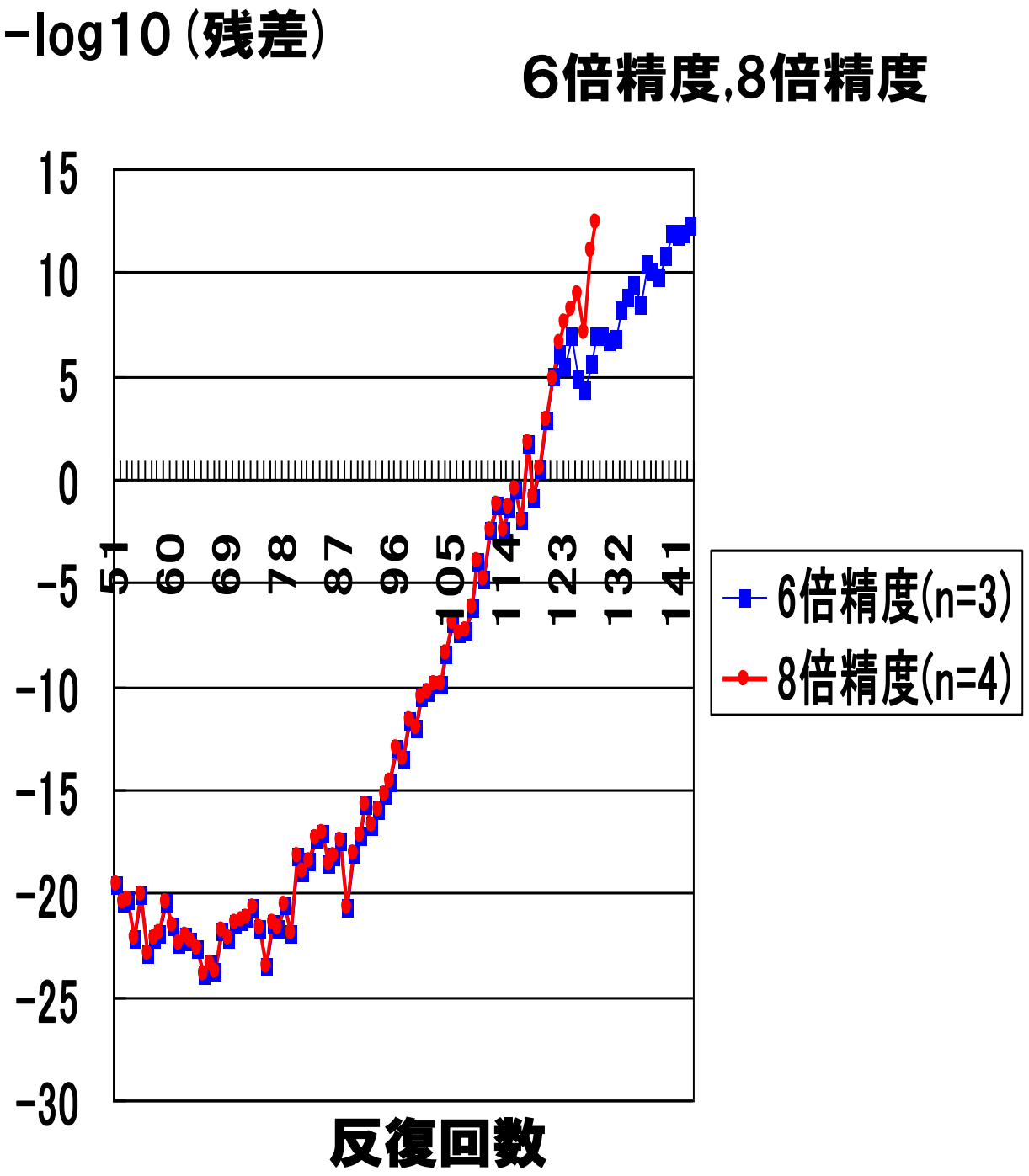

-log10(残差)
15
10
5
0
-5
-10
1
17
33
49
65
81
97
113
129
145
161
177
反復回数
4倍精度
5倍精度
6倍精度
7倍精度
8倍精度

CGS法の収束状況
nx=ny=nz=65 収束判定値=1.0q-12
倍精度演算では収束しない。



3.3.1.2　　CGS法の収束状況




-log10(残差)
6倍精度,8倍精度
15
10
5
0
-5
-10
-15
-20
-25
-30
51
60
69
78
87
96
105
114
123
132
141
反復回数
6倍精度(n=3)
8倍精度(n=4)

cgs法反復回数

倍精度は10000回で収束せず。

反復回数

精度	ieee	ieee smp	dd
4倍精度	392	448	670
5倍精度	226	155	none
6倍精度	130	131	144
7倍精度	128	128	none
8倍精度	128	128	128

q4sum,q5sum 使用

精度	反復回数
4倍精度	378
5倍精度	155

(1) ieee形式,dd形式ともに6倍精度演算が最も効率が良い事を示しています。

(2) q4sum,q5sumの様に総和演算を無限精度演算を使用すると,5倍精度の収束が良くなり,ieeeでのsmp実行と同じ反復回数となっています。

詳細な収束状況

dd形式cgs反復回数

ビット数	反復回数
95	4931
96	2155
97	3506
98	2811
99	1250
100	1103
101	1080
102	1027
103	815
104	702
105	630
106	670

3.3.2 サイズを拡大した場合

通常サイズを大きくすると収束に要する反復回数は増えますが、その状況を以下の条件で検証しました。

解法

bcg(biconjugate gradient)**法**

$R = 100, nx = ny = nz = 129$

cgs(conjugate gradient square)**法**

$R = 100, nx = ny = nz = 113$

収束判定値は共役残差 10^{-12}

初期値 $u(x, y, z) = 0$

の場合

反復回数10000まででは、
bcg法は倍精度演算,
cgs法は4倍精度演算まで収束しません。

bcg法反復回数一覧

演算精度	SR16000	x5570	E5-2670	Phi5110P
3倍精度		482		
4倍精度	316	308	308	314
5倍精度		317		
6倍精度		317		
7倍精度		317		
8倍精度		317		

cgs法反復回数一覧表

演算精度	SR16000	x5570	E5-2670	Phi5110P
5倍精度		373	368	368
6倍精度	281	237	240	240
7倍精度		217	217	217
8倍精度	217	217	217	217

(注)

1.赤字はその計算機で最も性能がよかったもの。

2.6倍精度の有効ビット数はSR16000は
159ビット,他は177ビット

3.E5－2670,Phi5110Pでは内積部分に
無限精度演算を適 用しています。

bcg法反復回数

精度	反復回数	正規化	実測/正規化
56ビット	50001	none	none
60ビット	4315	node	none
64ビット	1579	931	1.696
68ビット	938	826	1.136
72ビット	784	738	1.062
76ビット	593	664	0.893
3倍精度	482	600	0.803
84ビット	439	545	0.806
88ビット	396	497	0.797
92ビット	368	455	0.809
96ビット	336	418	0.804
100ビット	325	386	0.842
104ビット	319	357	0.894
108ビット	313	331	0.946
4倍精度	308	308	1

(注)50001回は50000回反復
で収束しなかった事を示しています。

正規化
P有効ビット数=精度のビット数+1

正規化反復回数

$$308 \times \left(\frac{113}{P}\right)^2$$

性能的には3倍精度が最も効率が良いと言えます。

3.3.3 精度改善例

以下の様に精度改善を行う事により,
これまでのサイズおよびサイズをさらに
拡大しても4倍精度演算で収束するように
なりました。

前処理付きcgs**法により,並列処理で精度改善**

cgs(conjugate gradient square)**法**

$R = 100, nx = ny = nz = 129$

収束判定値は共役残差 10^{-12}

初期値 $u(x, y, z) = 0$

4**倍精度演算**

CGS (PARALLEL)

SIZE NX, NY, NZ = 129 129 129

EPS = 1.00000000E - 0012

ITERATION = 142

3.4 ファインマンループ積分

ファインマンループ積分の一般式は以下の様になる。

$$I(\varepsilon)=(-1)^{N}\left(\frac{1}{4\pi}\right)^{nL/2}\Gamma(N-nL/2)\int_{0}^{1}\prod_{i=1}^{N}dx_{i}\,\delta(1-x_{1}-x_{2}-.....-x_{N})\frac{C^{N-n(L+1)/2}}{(D-i\varepsilon C)^{N-nL/2}}$$

n : **時空次元** $(n=4)$, N : **ループ内の素粒子の数**, L : **ループの多重度**

ε : **実定数**, C, D : $x_1, x_2, ..., x_N$ **の多項式**

演算精度が問題となるのは,以下の例の様に仮想光子λが含まれる場合である。

(1) Infra vtx

$$I(s)=\int_{0}^{1}\int_{0}^{1-x}\frac{1}{D}dydx \quad (D=-xys+(x+y)^{2}m_{e}^{\ 2}+(1-x-y)\lambda^{2})$$

(2) Infra box

$$I(s)=\int_{0}^{1}\int_{0}^{1-x}\int_{0}^{1-x-y}\frac{1}{D^{2}}\,dzdydx$$

$$(D=-xys-tz(1-x-y-z)+(x+y)\lambda^{2}$$

$$+(1-x-y-z)(1-x-y)m_{e}^{\ 2}+z(1-x-y)m_{f}^{\ 2})$$

３．４．１数値積分法

数値積分法としては、

(1) 誤差を評価しながら領域を二分割して行く
Gauss-Kronrod積分法

(2) 端点特異点を持つ被積分関数の積分に
強く,基本的には領域は分割せず，標本点を
ふやして積分計算結果の精度をあげていく
二重指数関数型積分法がある。

これ以外にも一重指数関数型積分法もあるが,
二重指数関数型積分法に比べ標本点数を大きく
とる必要がある。

端点と領域内部で同時に特異点をもつ
場合の計算を，sin c求積法によるHadamard
有限部分計算で行う方法もあるがファインマン
ループ積分への適用出来る例は少ない。

数値積分で積分領域内に特異点がある場合
微小量εを使用してD－＞D＋iεと有理化して
ε－＞0で得た値を求める積分値とする場合に
加速法を使用するが主にRicardson加速法,
エイトケン加速法、Wynnのε－算法等を用いる。

Gauss – Kronrod積分法と二重指数関数 型積分法
の選択に関しては以下 の差があります。
分点と重み係数のテーブル作成方法では
Gauss – Kronrod法では, まずGauss – 積分法で
$2n+1$の分点と重み係数を求めます。

$$-1 < x_1 < x_2 < .... < x_n < x_{n+1} = 0 < x_{n+2} < x_{n+3} < ... < x_{2n+1} < 1$$

これに$2n+2$の分点を追加して計 $4n+3$個の分点と重み係数
を求めます。これは反 復法により求めるため, 積分計算の
前に分点と重み係数の 計算が必要となります 。
これに対し, 二重指数関数型積分は 分点と重み係数は
関数計算$(\sinh, \cosh, \exp)$で求まるため, 積分計算のプログラム
に入れる事が可能です 。二重指数関数型積分 の誤差評価
は, $N = 2n+1$で求めた値$w_1, w_2, ..., w_{2n+1}$で

$$I_1 = h \times \sum_{i=1}^{2n+1} w_i$$ と $$I_2 = 2h \times \sum_{i=1}^{n+1} w_{2i-1}$$ を比較して求められますし,

$$\text{誤差} = \exp(-c_2 \frac{N}{\log(N)})$$ で評価ができます。

内部特異点がある場合 は領域分割によって計 算が可能です。

幾つかの例を次ページに記しました。

例題

4倍精度演算,分点数496,**変数変換区間**$[-1,1]$

で計算.分母d**で**$d=0$**となる点で領域を分割し,**

それぞれの領域で$d->d+i\varepsilon(\varepsilon_0=1.2^{-30},....,\varepsilon_{14}=1.2^{-44})$

$\varepsilon-$**算法で計算**

$$\int_{-1}^{1}\frac{x}{(x-\frac{1}{e})}dx=3.141592653589793$$

計算結果 $=3.141592653589793$

$$\int_{0}^{\frac{\pi}{2}}\frac{x}{(\sin x-2\cos x)^2}dx=-0.7669479668299477$$

計算結果 $=-0.7669479668299477$

$$\int_{0}^{\frac{\pi}{2}}\frac{x\sin 2x}{1-2\times 2.0\cos 2x+2.0^2}dx=0.1592257756149500$$

計算結果 $=0.1592257756149501$

$$\int_{0}^{1}\frac{1}{\log|\log(x)|}dx=-0.1544796413 20$$

計算結果 $=-0.154479641320$

(注)この積分値は約200**年間**0**と考えられていました。**

$$I=\int_{-1}^{1} f(x)dx \qquad f(x)=\frac{1}{(2-x)(1+x)^{\frac{3}{4}}(1-x)^{\frac{1}{4}}}$$

$h=0.2, N=40 \quad 10^{-13}$ **の相対誤差**

$$C_2=2.76 \quad \exp(-C_2\frac{N}{\ln(N)})$$

$$f(x)=\frac{(1+x)^{\frac{1}{4}}(1-x)^{\frac{3}{4}}}{(2-x)(1+x)(1-x)}=-\frac{1}{3}\frac{(1+x)^{\frac{1}{4}}(1-x)^{\frac{3}{4}}}{2-x}$$

$$+\frac{1}{6}(1+x)^{-\frac{3}{4}}(1-x)^{\frac{3}{4}}+\frac{1}{2}(1+x)^{\frac{1}{4}}(1-x)^{-\frac{1}{4}}$$

$$y=\frac{1}{2}(x+1) \qquad dx=2dy \qquad x+1=2y, 1-x=2(1-y)$$

$$I=\int_{-1}^{1} f(x)dx=-\frac{4}{3}\int_0^1 \frac{y^{\frac{1}{4}}(1-y)^{\frac{3}{4}}}{3-2y}dy+\frac{1}{3}\int_0^1 y^{-\frac{3}{4}}(1-y)^{\frac{3}{4}}dy+\int_0^1 y^{\frac{1}{4}}(1-y)^{-\frac{1}{4}}dy$$

$$=-\frac{4}{3}\int_0^1 \frac{y^{\frac{1}{4}}(1-y)^{\frac{3}{4}}}{3-2y}dy+\frac{1}{2}\Gamma(\frac{1}{4})\Gamma(\frac{3}{4})$$

$$J=\int_0^1 \frac{y^{\frac{1}{4}}(1-y)^{\frac{3}{4}}}{3-2y}dy=\frac{1}{2}\int_0^1 y^{\frac{1}{4}}(1-y)^{-\frac{1}{4}}dy+\frac{1}{4}\int_0^1 y^{-\frac{3}{4}}(1-y)^{-\frac{1}{4}}dy$$

$$-\frac{3}{4}\int_0^1 \frac{y^{-\frac{3}{4}}(1-y)^{-\frac{1}{4}}}{3-2y}dy=(\frac{1}{8}+\frac{1}{4}-\frac{\sqrt[4]{3}}{4})\Gamma(\frac{1}{4})\Gamma(\frac{3}{4})$$

$$I=[\frac{1}{2}-\frac{4}{3}(\frac{3}{8}-\frac{\sqrt[4]{3}}{4})]\Gamma(\frac{1}{4})\Gamma(\frac{3}{4})=\frac{\sqrt[4]{12}}{3}\pi=1.949054259166747$$

実測値(4**倍精度演算**) $=1.949054259166746$

被積分関数が内部特異点を持つ場合，D->D-iεで有理化して，ε->0として積分値を求めるのが一般的であるが収束が遅くなったり，異なる積分値を得る場合がある。

(1) inf ra　　vertex

$$\int_0^1 \int_0^{1-x} \frac{1}{D} dydx \quad (D = -sxy + (x+y)^2 m^2 + (1-x-y)\lambda^2)$$

では

$s < 0$**の場合の解析近似解は**

$$(\frac{1}{-s})[\ln(\frac{\lambda^2}{m_e{}^2})\ln(\frac{m_e{}^2}{-s}) + \frac{1}{2}\ln^2(\frac{m_e{}^2}{-s}) - \frac{\pi^2}{6}]$$

で$s > 0$**のときは**，$s- > s + i\varepsilon$**で解析接続して求める。**

$s = 500^2, m = 0.0005$**で**Gauss – Kronrod**法**

4倍精度演算で計算した時の実行時間(秒)は以下の様になっています。

$D- > D - i\varepsilon \qquad e = \varepsilon$

$s- > s + i\varepsilon \qquad e = -xy\varepsilon$

λ	$e = \epsilon$	$e = -xy\epsilon$	性能向上比
10^{-9}	1006.713	505.57	1.99
10^{-12}	1819.767	542.21	3.36
10^{-15}	2684.821	505.80	5.31
10^{-20}	3254.224	522.48	6.22

s− > s + iε**で計算**

$\lambda = 10^{-30}$から 10^{-240}までの計算

実数部演算結果

λ	**演算精度**	**解析値**	**実測値**
10^{-30}	8 倍精度	-0.150899286980482291 5D-01	-0.15089928698076975 3D-01
10^{-60}	16 倍精度	-0.303593952562260154 2D-01	-0.30359395256285495 1D-01
10^{-80}	16 倍精度	-0.405390396283445396 0D-01	-0.40539039628423507 5D-01
10^{-160}	32 倍精度	-0.812576171168186363 23D-01	-0.81257617075281066 6D-01
10^{-200}	32 倍精度	-0.101616905861055684 68	-0.10161690585744838 8
10^{-240}	32 倍精度	-0.121976194605292733 04	-0.12197619460096628 8

実数部演算結果

λ	**演算精度**	**解析値**	**実測値**
10^{-30}	8 倍精度	-0.150899286980482291 5D-01	-0.15089928698076975 3D-01
10^{-60}	16 倍精度	-0.303593952562260154 2D-01	-0.30359395256285495 1D-01
10^{-80}	16 倍精度	-0.405390396283445396 0D-01	-0.40539039628423507 5D-01
10^{-160}	32 倍精度	-0.812576171168186363 23D-01	-0.81257617075281066 6D-01
10^{-200}	32 倍精度	-0.101616905861055684 68	-0.10161690585744838 8
10^{-240}	32 倍精度	-0.121976194605292733 04	-0.12197619460096628 8

非常に良い精度の結果が得られている。

(2) inf ra box

$$I = \int_0^1 \int_0^{1-x} \int_0^{1-x-y} \frac{1}{D^2} dz dy dx$$

>>

$$D = -xys - tz(1-x-y-x) + (x+y)\lambda^2 + (1-x-y-z)(1-x-y)m_e{}^2 + z(1-x-y)m_f{}^2$$

で$s < 0$**の場合の解析近似解**

$$I = \frac{1}{-s(-t+m_f{}^2)} \ln(\frac{-s}{\lambda^2}) \ln \frac{(-t+m_f{}^2)^2}{m_e{}^2 m_f{}^2}$$

から$s > 0$**の場合**$s- > s+i\varepsilon$**と解析接続して計算します。**

$$I = \int_0^1 \int_0^{1-x} \int_0^{1-x-y} \frac{1}{D^2} dz dy dx$$ **で**$D- > D-i\varepsilon$**で有理化して得た虚数 部の値は**ε**の初期値**ε_0**の取り方により結果が 大きく異なる。**

(数列としては十分な精 度で収束している。)

(1) $\varepsilon_0 >> m_e{}^2$	0.14264757279476114 3d - 09
(2) $m_e{}^2 >> \varepsilon_0 >> \lambda^2$	0.37153689286759225 0d - 08
(3) $\lambda^2 >> \varepsilon_0$	0.74307378490002147 2d - 08

収束の速さでは圧倒的に(1)が速い。

これに対し,$s- > s+i\varepsilon$では(1)と同じ収束の速さで(3)の結果が得られる。

ループ積分で注意すべき点は,積分値が$\log$の関数
を含んだ場合である。
例えば,

$$\int_0^1 \frac{1}{M^2 x + (1-x)\lambda^2} dx = \frac{1}{M^2 - \lambda^2} \log(\frac{M^2}{\lambda^2})$$で

$\lambda^2 = 0$とすると数学上では∞で,プログラム実行時に
エラーが発生すると考えられる。ところが$\lambda^2 = 0$としても
プログラムは正常に終了し,それらしい結果が得られる。
数値積分法で広義積分と結果が一致するのは広義積分
が有限値を持つ場合である。二重指数関数型積分法で

$$\int_0^1 \frac{1}{x} dx$$の$z =$ 重み$\times$被積分関数の値は

$$z = (1-x)\sqrt{\log(\frac{x}{1-x})^2 + \pi^2}$$で$0 < x$でもっとも小さい値をε
とすると,この値は$\log(\frac{1}{\varepsilon})$となる。この値は大きい様に
見えるが,倍精度$\varepsilon = 10^{-308}$なら$\log(\frac{1}{\varepsilon}) \fallingdotseq 709$
4倍精度$\varepsilon = 10^{-4931}$なら$\log(\frac{1}{\varepsilon}) \fallingdotseq 11354$で例題で$M = 100$
なら,倍精度演算で0.0709,4倍精度演算で1.1354とエラーが
発生する様な値にはならない。

二重指数型積分法では, $x = \phi(t)$

$|f(\phi(t))\phi'(t)| \approx \exp(-c\exp|t|), |t| \to \infty$の様に

減衰するとき最適の公式が得られる。

変数変換区間[0,1]でのDEでは,

$y = \frac{\pi}{2}\sinh(t), x = \frac{e^{y}}{e^{y}+e^{-y}}, 1-x = \frac{e^{-y}}{e^{y}+e^{-y}}$ より

$x(1-x) = \frac{1}{(e^{y}+e^{-y})^{2}}, w = \phi'(t) = \frac{\pi \times \cosh(t)}{(e^{y}+e^{-y})^{2}}$

$\frac{x}{1-x} = e^{2y}, 2y = \log(\frac{x}{1-x}),$

$\pi \times \cosh(t) = \pi \times \sqrt{\sinh(t)^{2}+1} = \pi \times \sqrt{(\frac{2}{\pi}y)^{2}+1}$

$= \pi \times \sqrt{\frac{4y^{2}+\pi^{2}}{\pi^{2}}} = \sqrt{4y^{2}+\pi^{2}}$

$= \sqrt{(\log(\frac{x}{1-x}))^{2}+\pi^{2}}$

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

このため被積分関数$f(x)$の[0,1]における積分で

DEが効果があるかどうかは$x = \varepsilon, 1-\varepsilon (0 < \varepsilon << 1)$での

$f(x)x(1-x)\sqrt{(\log(\frac{x}{1-x}))^{2}+\pi^{2}}$が十分減衰しているか

どうかを見れば良い。

Beta関数

$\int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx \quad (\alpha>0,\beta>0)$では,

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

$$x = \frac{1}{2} \qquad I = \pi(\frac{1}{2})^{\alpha+\beta} \cong \pi \quad (0<\alpha,\beta<<1)$$

$$x = \varepsilon \qquad I = \sqrt{(\log(\varepsilon)^2 + \pi^2}\,\varepsilon^{\alpha} \cong \varepsilon^{\alpha}\log(\frac{1}{\varepsilon})$$

$$x = 1-\varepsilon \qquad I \cong \varepsilon^{\beta}\log(\frac{1}{\varepsilon})$$

一般に, $f(x) = x^{\alpha}\log x \quad (0<\alpha<1, 0<x<1)$

$x = 10^{-n}$とおくと, $|f(x)| = \frac{n\log 10}{10^{n\alpha}}$

$|f(x)| = 10^{-m}$ (mは得たい精度を表わす数)となる

nは$n > \frac{m}{\alpha}$。 $m = 10, \alpha = 0.01$なら$n > 1000$より

指数部は15ビットが必要になる事がわかる。また

$x = 1-\varepsilon$でのβから, $\varepsilon = 10^{-1000}$が必要となり, 指数部15ビットの変数を2つつなげた演算が必要になる事が容易にわかる。

具体的計算例

(1) inf ra　　vtx

変数変換後の f は

$$f = \frac{x}{-sx^2y(1-y) + x^2 m_e{}^2 + (1-x)\lambda^2} \quad (s = -500^2, m_e = 0.0005)$$

これから $|f(\phi(t_1),\phi(t_2))\phi'(t_1)\phi'(t_2)|$ **を求めると**,

$x = \varepsilon, y = \varepsilon, 1-\varepsilon$

で $\frac{\varepsilon^3 (\log(\frac{1}{\varepsilon}))^2}{-s\varepsilon^3 + \varepsilon^2 m_e{}^2 + \lambda^2} \fallingdotseq 1.51\times 10^{-17} \quad \lambda = 10^{-30}, \varepsilon = \frac{\lambda}{m_e} = 2\times 10^{-27}$

$x = 1-\varepsilon, y = \varepsilon, 1-\varepsilon$

で $\frac{\varepsilon^2 (\log(\frac{1}{\varepsilon}))^2}{-s\varepsilon + m_e{}^2 + \varepsilon\lambda^2} \fallingdotseq 1.53\times 10^{-15} \quad \varepsilon = \frac{m_e{}^2}{-s} = 10^{-12}$

これより, $x = \varepsilon, y = 1-\varepsilon$ **に重点を置けば良い事がわかる。**

(2) inf ra　　box

変数変換後の f は

$$f = \frac{x(1-x)}{D^2}$$

$$D = -sx^2y(1-y) + x\lambda^2 + (1-x)^2 A(z)$$

$$A(z) = tz^2 + (-t + m_f{}^2 - m_e{}^2)z + m_e{}^2$$

$$(s = -500^2, t = -150^2, m_f = 150, m_e = 0.0005)$$

$y = \varepsilon, 1-\varepsilon$ **で**, $|f(\phi(t_1),\phi(t_2),\phi(t_3))\phi'(t_1)\phi'(t_2)\phi'(t_3)|$ **は**

$$\frac{x^2(1-x)^2 z(1-z)\sqrt{(\log(\frac{x}{1-x}))^2 + \pi^2}\sqrt{(\log(\frac{z}{1-z}))^2 + \pi^2}\,\varepsilon\log(\frac{1}{\varepsilon})}{D^2}$$

$$D = -sx^2\varepsilon + x\lambda^2 + (1-x)^2 A(z)$$

$z=\varepsilon$のとき$A(z)={m_e}^2, z=1-\varepsilon$のとき$A(z)={m_f}^2$から

$z=\varepsilon$のときが最大値

$$\frac{x^2(1-x)^2\sqrt{(\log(\frac{x}{1-x}))^2+\pi^2\varepsilon^2(\log(\frac{1}{\varepsilon}))^2}}{D^2}$$

$D=-sx^2\varepsilon+x\lambda^2+(1-x)^2{m_e}$となる。

$x=\varepsilon,1-\varepsilon$ともに分子は同じで

$$\frac{\varepsilon^4(\log(\frac{1}{\varepsilon}))^3}{D^2}$$

$x=\varepsilon \quad D=-s\varepsilon^3+\varepsilon\lambda^2+{m_e}^2\geq{m_e}^2$

$x=1-\varepsilon \quad D=-s\varepsilon+\lambda^2+\varepsilon^2{m_e}^2\geq\lambda^2$

これより

$z=\varepsilon, x=1-\varepsilon$の場合の$\dfrac{\varepsilon^4(\log(\frac{1}{\varepsilon}))^3}{\lambda^4}$を考えれば良い。

$\lambda=10^{-m}, \varepsilon=10^{-n}\ \lambda=10^{-(n+m)}$とすると,

$$\frac{\varepsilon^4(\log(\frac{1}{\varepsilon}))^3}{\lambda^4}=10^{-4n}((m+n)\log 10)^3$$

$m=30,n=3$だと$4.387\times10^{-7} \quad n=2.9$だと$1.09\times10^{-6}$

これは,10進6桁の精度を得るには$1-\varepsilon$で$\varepsilon=2^{-109}$が必要と示している.これは4倍精度でieee754-2008形式では10進6桁の精度の結果が得られ,DD形式では1に近い数の表現に注意しないと10進6桁の精度が得られず,1に近い数の表現に注意すると,ieee754-2008形式よりも精度の良い結果が得られる事と良く対応している。

3.4.2 massless計算

$$a^x = 1 + (\log(a))x + \frac{(\log(a))^2}{2!}x^2 + \cdots\cdots \quad (a > 0, |x| < \infty)$$

ここで,$|x|$**が小として一次の項まで取って**

$a^x = 1 + (\log(a))x$**とする**.

$$\frac{1}{a}\log(\frac{a}{c}) = \frac{a^{\eta-1}}{\eta} \qquad \eta = \frac{1}{\log(\frac{1}{c})}$$

$$\frac{1}{a}\log(a) + \frac{1}{a}\log(\frac{1}{c}) = \frac{\log(a)}{a} + \frac{1}{a\eta} = \frac{1 + (\log(a))\eta}{a\eta} = \frac{a^{\eta-1}}{\eta}$$

(1)vtx

$$\int_0^1 \int_0^{1-x} \frac{1}{-sxy + (x+y)^2 m^2 + (1-x-y)\lambda^2} dydx$$

$s < 0, -s >> m^2 >> \lambda^2, \frac{m^2}{-s} = e, \frac{\lambda^2}{-s} = g$**と正規化して**,

$$I = \int_0^1 \int_0^{1-x} \frac{1}{xy + (x+y)^2 e + (1-x-y)g} dydx$$

$$= \int_0^1 \int_0^1 \frac{p}{p^2 q(1-q) + p^2 e + (1-p)g} dpdq$$

$D = g^2 - 4g(q(1-q)+e) < 0$**より**

$$I = \int_0^1 \frac{1}{2A}\log(\frac{A}{g})dq \qquad A = q(1-q) + e$$

$$= \frac{1}{2\eta}\int_0^1 A^{\eta-1}dq \qquad \eta = \frac{1}{\log(\frac{1}{g})}$$

$$J=\int_0^1\int_0^{1-x}\frac{1}{(xy+(x+y)^2e)^{1-\eta}}dydx=\int_0^1\int_0^1\frac{p}{p^{2-\eta}A^{1-\eta}}dpdq=\frac{1}{2\eta}\int_0^1 A^{\eta-1}dq$$

よって$I=J$

$$I=J=\frac{1}{2}\int_0^1\frac{\log(A)}{A}dq+\frac{1}{2\eta}\int_0^1\frac{1}{A}dq \qquad \log(\frac{1}{\lambda^2})$$**の一次式**

$m^2=0,e=0$**とすると**

$$I=\int_0^1 q^{\eta-1}(1-q)^{\eta-1}dq=\frac{1}{2\eta}B(\eta,\eta)=\frac{1}{\eta^2}\frac{(\Gamma(1+\eta))^2}{\Gamma(1+2\eta)}$$

$\log(\frac{1}{\lambda^2})$**の二次式**

一般公式

$$0<t<<1 \qquad \int_0^1\frac{1}{-x^2+x+t}dx=2\log(\frac{1}{t})$$

$$a<0,0<c<<1$$

$$\int_0^1\frac{\log(\frac{-ax(1-x)+c}{c})}{-ax(1-x)+c}dx=\frac{1}{-a}[\log^2(\frac{c}{-a})-\frac{\pi^2}{3}]$$

$s=-500^2,m=0.0005,\lambda=10^{-2500}$

解析近似解　1.272295747

計算解　1.272296056

$s=500^2,m=0.0005,\lambda=10^{-2500}$

解析近似解　**実数 部**　$-1..272276008$

虚数部　0.1448318783

計算解　**実 数部**　$-1..272276295$

虚数部　0.1448319591

$s = 500^2, m = 0.0005$

λの値が充分小さくないと結果の精度が良くない事があります。

実 測値と解析式の相対誤 差

λ	実数 部	虚数部
10^{-15}	6.23×10^{-3}	1.38×10^{-2}
10^{-150}	6.30×10^{-5}	1.49×10^{-4}
10^{-1500}	6.68×10^{-7}	1.59×10^{-6}
10^{-2500}	2.26×10^{-7}	5.58×10^{-7}
10^{-15000}	6.42×10^{-9}	1.61×10^{-8}

$m^2 = 0, e = 0$**とすると**

$$\int_0^1 \int_0^{1-x} (xy)^{\eta-1} dydx = \frac{1}{\eta}\int_0^1 x^{\eta-1}(1-x)^{\eta} dx = \frac{1}{\eta^2}\frac{(\Gamma(1+\eta))^2}{\Gamma(1+2\eta)}$$

$I = \int_0^1 x^{\eta-1}(1-x)^{\eta} dx$**で**

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

$$x = \frac{1}{2} \quad |f(\phi(t))\phi'(t)| \cong \frac{\pi}{2}$$

$$x = 1-\varepsilon \quad |f(\phi(t))\phi'(t)| \cong \log(\frac{1}{\varepsilon})\varepsilon^{\eta+1}$$

$$\eta = 0.01 \quad \varepsilon = 10^{-12} \quad |f(\phi(t))\phi'(t)| \cong 2.5\times10^{-11}$$

$$x = \varepsilon \quad |f(\phi(t))\phi'(t)| \cong \log(\frac{1}{\varepsilon})\varepsilon^{\eta}$$

$$\eta = 0.01 \quad \varepsilon = 10^{-300} \quad |f(\phi(t))\phi'(t)| \cong 0.691$$

$$\varepsilon = 10^{-2450} \quad |f(\phi(t))\phi'(t)| \cong 2\times10^{-21}$$

計算結果

case1
η=0.01 λ=1.928749847963918E-022
解析解 9998.378865794316870325682352395056 1
倍精度 9328.03614724728　　N=1024
倍精度(改)　9998.37886579432 N=1024
拡張倍精度　9998.37886579431643 N=738
4倍精度(ieee) 9998.378865794316870325678679367 06　　N=772
4倍精度(dd)　9362.376631779167369581189462115687 5　N=1184

case2
η=7.238241365054197E-003 λ=1.000000000000000E-030
解析解　19085.205538478496689369143315971833
倍精度　15919.3830485798　　N=1024
倍精度(改) 19085.2055384785　　N=1024
拡張倍精度 19085.2055384784966e N=738
4倍精度(ieee) 19085.205538478496641497150058927 6　N=772
4倍精度(dd)　16034.373970898940278247867121568078 N=1184

解析結果

$$\int_0^1\int_0^{1-x}\frac{1}{(xy)^{1-\eta}}dydx=\frac{1}{\eta^2}\frac{(\Gamma(1+\eta))^2}{\Gamma(1+2\eta)}$$

$$\int_0^1\int_0^{1-x}\frac{1}{(xy)^{1-\eta}}dydx=\frac{1}{\eta}\int_0^1 x^{\eta-1}(1-x)^{\eta}dx$$

$f(x)=x^{\eta-1}(1-x)^{\eta}$**とすると**

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

$x=\varepsilon,|f(\phi(t))\phi'(t)|\cong\varepsilon^{\eta}\log(\frac{1}{\varepsilon})$**となり,表現可能な**$\varepsilon$**の値に精度が依存する事になる。**

$$\int_0^1\int_0^{1-x}(xy)^{\eta}dydx=\frac{1}{\eta+1}\int_0^1 x^{\eta}(1-x)^{\eta+1}dx=\frac{(\Gamma(1+\eta))^2}{(1+2\eta)(2+2\eta)\Gamma(1+2\eta)}$$

より$\int_0^1\int_0^{1-x}\frac{1}{(xy)^{1-\eta}}dydx=\frac{(1+2\eta)(2+2\eta)}{\eta^2}\int_0^1\int_0^{1-x}(xy)^{\eta}dydx$

$\int_0^1\int_0^{1-x}(xy)^{\eta}dydx$**は**$x=p(1-q),y=pq$**で変数変換して,**

$$\int_0^1\int_0^{1-x}(xy)^{\eta}dydx=\int_0^1 P^{1+2\eta}dp\times\int_0^1 q^{\eta}(1-q)^{\eta}dq$$

$\int_0^1 P^{1+2\eta}dp$**では,**$|f(\phi(t))\phi'(t)|=\sqrt{(\log(\frac{x}{1-x}))^2+\pi^2}\,x^{2+\eta}(1-x)$

$x=\varepsilon,1-\varepsilon$**で** $|f(\phi(t))\phi'(t)|$ **は十分小さくなる。**

$\int_0^1 q^{\eta}(1-q)^{\eta}dq$**では,**$x=\varepsilon,1-\varepsilon$**で** $|f(\phi(t))\phi'(t)|\cong\varepsilon^{\eta+1}\log(\frac{1}{\varepsilon})$

となり,倍精度変数でε**の値は取れる。**

一般化

$$\int_0^1 \int_0^{1-x} \cdots\cdots \int_0^{1-x_1-\cdots\cdots x_{n-1}} \frac{1}{(x_1 x_2 \cdots\cdots x_n)^{1-\eta}} dx_n dx_{n-1} \cdots\cdots dx_1$$

$$= \frac{1}{\eta^n} \frac{(\Gamma(1+\eta))^n}{\Gamma(1+n\eta)}$$

$$\frac{(\Gamma(1+\eta))^n}{\Gamma(1+n\eta)} = \frac{1+n\eta}{(1+\eta)^n} \exp[\sum_{m=2}^{\infty} (-1)^m \frac{\varsigma(m)-1}{m} (n-n^m)\eta^m]$$

$\frac{(\Gamma(1+\eta))^n}{\Gamma(1+n\eta)}$**は以下の様にして求める事ができる。**

$$a_m = (-1)^m \frac{n - n^m}{m} \varsigma(m)$$

$$f_0 = 1, f_1 = 0, f_2 = a_2, f_3 = a_3, f_4 = a_4 + \frac{1}{2} {a_2}^2, f_5 = a_5 + a_2 a_3$$

$$f_6 = a_6 + a_2 a_4 + \frac{1}{2} {a_3}^2 + \frac{1}{6} {a_2}^3$$

$$f_7 = a_7 + a_2 a_5 + a_3 a_4 + \frac{1}{2} {a_2}^2 a_3$$

$$f_8 = a_8 + a_2 a_6 + a_3 a_5 + \frac{1}{2} {a_4}^2 + \frac{1}{2} {a_2}^2 a_4 + \frac{1}{2} a_2 {a_3}^2 + \frac{1}{24} {a_2}^4$$

$$n = 5$$

$$a_2 = -10\zeta(2), a_3 = 40\zeta(3), a_4 = -155\zeta(4)$$

$$a_5 = 624\zeta(5)$$

展開式

$$1 - 10\zeta(2)\eta^2 + 40\zeta(3)\eta^3$$

$$+ (-155\zeta(4) + 50(\zeta(2))^2)\eta^4$$

$$+ (624\zeta(5) - 400\zeta(2)\zeta(3))\eta^5$$

$$n = 6$$

$$a_2 = -15\zeta(2), a_3 = 70\zeta(3), a_4 = -\frac{645}{2}\zeta(4)$$

$$a_5 = 1554\zeta(5), a_6 = -7775\zeta(6)$$

展開式

$$1 - 15\zeta(2)\eta^2 + 70\zeta(3)\eta^3$$

$$+ (-\frac{645}{2}\zeta(4) + \frac{225}{2}(\zeta(2))^2)\eta^4$$

$$+ (1554\zeta(5) - 1050\zeta(2)\zeta(3))\eta^5$$

$$+ (-7775\zeta(6) + \frac{9675}{2}\zeta(2)\zeta(4) + 2450(\zeta(3))^2$$

$$- \frac{1125}{2}(\zeta(2))^3)\eta^6$$

$$n = 7$$

$$a_2 = -21\varsigma(2), a_3 = 112\varsigma(3), a_4 = -\frac{1197}{2}\varsigma(4)$$

$$a_5 = 3360\varsigma(5), a_6 = -19607\varsigma(6), a_7 = 117648\varsigma(7)$$

展開式

$$1 - 21\varsigma(2)\eta^2 + 112\varsigma(3)\eta^3$$
$$+ (-\frac{1197}{2}\varsigma(4) + \frac{441}{2}(\varsigma(2))^2)\eta^4$$
$$+ (3360\varsigma(5) - 2352\varsigma(2)\varsigma(3))\eta^5$$
$$+ (-19607\varsigma(6) + \frac{25137}{2}\varsigma(2)\varsigma(4) + 6272(\varsigma(3))^2$$
$$- \frac{3087}{2}(\varsigma(2))^3)\eta^6$$
$$+ (117648\varsigma(7) - 70560\varsigma(2)\varsigma(5) - 67032\varsigma(3)\varsigma(4)$$
$$+ 24696(\varsigma(2))^2\varsigma(3))\eta^7$$

$$n = 8$$

$$a_2 = -28\varsigma(2), a_3 = 168\varsigma(3), a_4 = -1022\varsigma(4)$$

$$a_5 = 6552\varsigma(5), a_6 = -\frac{131068}{3}\varsigma(6), a_7 = 299592\varsigma(7)$$

$$a_8 = -2097151\varsigma(8)$$

展開式

$$1 - 28\varsigma(2)\eta^2 + 168\varsigma(3)\eta^3$$
$$+ (-1022\varsigma(4) + 392(\varsigma(2))^2)\eta^4$$
$$+ (6552\varsigma(5) - 4704\varsigma(2)\varsigma(3))\eta^5$$
$$+ (-\frac{131068}{3}\varsigma(6) + 28616\varsigma(2)\varsigma(4) + 14112(\varsigma(3))^2$$
$$- \frac{21952}{6}(\varsigma(2))^3)\eta^6$$
$$+ (299592\varsigma(7) - 183456\varsigma(2)\varsigma(5) - 171696\varsigma(3)\varsigma(4)$$
$$+ 65856(\varsigma(2))^2\varsigma(3))\eta^7$$
$$+ (-2097151\varsigma(8) + \frac{3669904}{3}\varsigma(2)\varsigma(6) + 110736\varsigma(3)\varsigma(5)$$
$$+ 522242(\varsigma(4))^2 - 400624(\varsigma(2))^2\varsigma(4) - 395136\varsigma(2)(\varsigma(3))^2$$
$$+ \frac{76382}{3}(\varsigma(2))^4)\eta^8$$

$$\varsigma(2)=\frac{\pi^2}{6},\varsigma(4)=\frac{\pi^4}{90},\varsigma(6)=\frac{\pi^6}{945},\varsigma(8)=\frac{\pi^8}{9450}$$

実測結果$\eta=0.01$

dim5 = 0.998402809124425953047924794752494

dim5 = 0.998402809640216913557719749246

dim6 = 0.997616251685321110907418255207708

dim6 = 0.997616251653673359393296190115

dim7 = 0.996679645316743857719951968389312

dim7 = 0.996679645317184684228686126118

dim8 = 0.995595435273981970066429062072135

dim8 = 0.995595435274204817137389673495

実測結果$\eta=0.001$

dim5 = 0.999983598708994068287163090479129 85

dim5 = 0.999983598708994592536696357810

dim6 = 0.999975410087871619637965820354303 88

dim6 = 0.999975410087871616448538275141

dim7 = 0.999965590962667639174415680604871 96

dim7 = 0.999965590962667639178603625416

dim8 = 0.999954143743733701838431902211522 45

dim8 = 0.999954143743733701839646669370

積分値の精度はηの値により表現可能な数値範囲に依存するため,この影響を少なくするために以下の公式を用いて計算する。

$$\int_0^1 \int_0^{1-x} \cdots\cdots \int_0^{1-x_1-\cdots\cdots x_{n-1}} \frac{1}{(x_1 x_2 \cdots\cdots x_n)^{1-\eta}} dx_n dx_{n-1} \cdots\cdots dx_1$$

$$= \frac{1}{\eta^n} \frac{(\Gamma(1+\eta))^n}{\Gamma(1+n\eta)}$$

$$\eta^n \int_0^1 \int_0^{1-x} \cdots\cdots \int_0^{1-x_1-\cdots\cdots x_{n-1}} \frac{1}{(x_1 x_2 \cdots\cdots x_n)^{1-\eta}} dx_n dx_{n-1} \cdots\cdots dx_1$$

$$= (1+n\eta)(2+n\eta)\cdots(n+n\eta) \int_0^1 \int_0^{1-x} \cdots\cdots \int_0^{1-x_1-\cdots\cdots x_{n-1}} (x_1 x_2 \cdots\cdots x_n)^{\eta} dx_n dx_{n-1} \cdots\cdots dx_1$$

一般ベータ関数

$$I_n = \int_{\Omega} x_1^{y_1-1} x_2^{y_2-1} \cdots\cdots x_n^{y_n-1} d\Omega \qquad (1)$$

$$J_n = \int_{\Omega} x_1^{y_1} x_2^{y_2} \cdots\cdots x_n^{y_n} d\Omega \qquad (2)$$

$$\Omega = \{x_i \geq 0 \mid \sum_{i=1}^{n+1} x_i = 1\}, d\Omega = dx_n dx_{n-1} \cdots\cdots dx_1$$

$$(y_1 y_2 \cdots\cdots y_n) I_n = (z_1 z_2 \cdots\cdots z_n) J_n \quad : \quad z_i = i + \sum_{j=1}^{n} y_j$$

(1)で計算する場合,4倍精度または4倍精度変数を2つつなげた8倍精度演算が必要。(DQ形式)

(2)で計算する場合は倍精 度またはDD形式の4倍精度演算で精度の良い結果が得られる。

(2)box

$$\eta = \frac{1}{\log(\frac{1}{\lambda^2})}$$

$$I = \int_0^1 \int_0^{1-x} \int_0^{1-x-y} \frac{1}{D^{2-\eta}} dzdydx$$

$$D = xz + y(1-x-y-z)$$

では,

$$I = \frac{1}{1-\eta}\frac{2}{\eta}(\frac{2}{\eta} - \frac{\pi^2}{4}2^{1-\eta}\eta)\frac{(\Gamma(1+\eta))^2}{\Gamma(1+2\eta)}$$

$$\frac{(\Gamma(1+\eta))^2}{\Gamma(1+2\eta)} = \frac{1+2\eta}{(1+\eta)^2}\exp[\sum_{k=2}^{\infty}(-1)^k(2-2^k)(\frac{\varsigma(k)-1}{k})\eta^k]$$

$$\log\Gamma(1+x) = -\log(1+x) + (1-\gamma)x + \sum_{k=2}^{\infty}(-1)^k(\frac{\varsigma(k)-1}{k})x^k$$ **より。**

$$I = \frac{a_{-2}}{\eta^2} + \frac{a_{-1}}{\eta} + a_0$$ **とすると**, $a_{-2} = 4, a_{-1} = 4, a_0 = -4(\frac{5}{12}\pi^2 - 1) = -\frac{5}{3}\pi^2 + 4$

これは以下の事からきています。

$$I = [\int_0^1 \frac{(1-q)^{1-\eta} - q^{1-\eta}}{(1-2q)q^{1-\eta}(1-q)^{1-\eta}}dq] \times \frac{1}{1-\eta}[\int_0^1 \frac{1}{p^{1-\eta}(1-p)^{1-\eta}}dp]$$

から $\frac{1}{1-\eta}[\int_0^1 \frac{1}{p^{1-\eta}(1-p)^{1-\eta}}dp] = \frac{1}{1-\eta}\frac{2}{\eta}\frac{(\Gamma(1+\eta))^2}{\Gamma(1+2\eta)}$ **。**

$\int_0^1 \frac{(1-q)^{1-\eta} - q^{1-\eta}}{(1-2q)q^{1-\eta}(1-q)^{1-\eta}}dq = \frac{2}{\eta} - \frac{\pi^2}{4}2^{1-\eta}\eta$ **となるがより詳細には**

次ぎの様になります。

$$\int_0^1 \frac{(1-q)^{1-\eta}-q^{1-\eta}}{(1-2q)q^{1-\eta}(1-q)^{1-\eta}}dq = \int_0^1 [\frac{q^{\eta}}{(1-2q)q} - \frac{(1-q)^{\eta}}{(1-2q)(1-q)}]dq$$

$$= 2\int_0^1 \frac{q^{\eta}-(1-q)^{\eta}}{1-2q}dq + \frac{2}{\eta} = \frac{2}{\eta} + 2^{1-\eta}\int_0^1 \frac{(1-t)^{\eta}-(1+t)^{\eta}}{t}dt$$

$$= \frac{2}{\eta} - 2^{1-\eta}2\eta(1+\sum_{k=1}^{\infty}\frac{(\eta-1)(\eta-2).....(\eta-2k)}{(2k+1)!(2k+1)})$$

( )**の定数項は**$-2(1+\sum_{k=1}^{\infty}\frac{1}{(2k+1)^2}) = -2(\sum_{k=1}^{\infty}\frac{1}{(2k-1)^2}) = -2\frac{3}{4}\varsigma(2)$

$$= -\frac{\pi^2}{4}$$

( )**の**η**の一次の項は負の数の和の絶対値で計算すると**

$$\sum_{k=1}^{\infty}\frac{\text{分子の項}}{(2k+1)!(2k+1)}.$$

分子の項$= (2k)!\sum_{r=1}^{2k}\frac{1}{r} < (2k)![\log(2k+1)+\gamma]$

$\gamma = 0.5772156649$ (**オイラーの定数**

$$\sum_{k=1}^{\infty}\frac{(2k)![\log(2k+1)+\gamma]}{(2k+1)!(2k+1)} = \sum_{k=1}^{\infty}\frac{\log(2k+1)}{(2k+1)^2} + \gamma\sum_{k=1}^{\infty}\frac{1}{(2k+1)^2}$$

$$\sum_{k=1}^{\infty}\frac{\log(2k+1)}{(2k+1)^2} = \pi^2[\frac{3}{2}\log A - \frac{1}{6}\log(2) - \frac{1}{8}\log(\pi) - \frac{\gamma}{8}] = 0.4181157384$$

$$A = \lim_{n->\infty}\frac{1\times 2^2\times 3^3\times .....\times n^n}{n^{n^2/2+n/2+1/12}}e^{\frac{n^2}{4}} = 1.282427130$$

$$\log A = 0.2487544703$$

$$\gamma\sum_{k=1}^{\infty}\frac{1}{(2k+1)^2} = \gamma[\sum_{k=1}^{\infty}\frac{1}{(2k-1)^2} - 1] = \gamma(\frac{\pi^2}{8} - 1) = 0.1348956184$$

これから( )**内の**η**の一次の項の係数は**-0.5530113568

全体では$-2^{1-\eta}\times 2\eta$**をかけて**$2.212\eta^2$**となりこれが一次式まで取った場合の誤差となる。**

実測例としては以下のものがあります。

解析近似式を使用して，$\lambda = 10^{-1000000001} \sim 10^{-1000000045}$

で計算しました。**ただしgamma関数はサポートされていない機種もあり,展開式より計算しています。**

a2= 4.00000000000000065094194454995095
a1= 4.00000000000001117756660514563709
a0= -12.44934066761187120664453064541111

変数変換区間$SR16000[10^{-100},1-10^{-100}]$, x5570**と**x2670**では**
$[10^{-200},1-2^{-113}]$**の4倍精度では**
epsilon**算法の収束具合は**
$\lambda = 10^{-150}$**で**

	値	**誤差**
x5570	0.191143398445070834D + 07	0.14063331231312 7879D - 05
x2670	0.191143398444577885D + 07	0.21027092538834 4072D - 07
sr16000	0.191143398444582948Q + 07	0.15665822292190 9210Q - 05

$\lambda = 10^{-1500}$**で**

x5570	0.190895950695064375D + 09	0.91950002332009 1177D - 01
x2670	0.190895950598091269D + 09	0.47781492985379 9862D - 01
sr16000	0.190895950686589134Q + 09	0.47081222571149 9829Q - 01

$\lambda = 10^{-15000}$**で**

x5570	0.190871021209837876D + 11	0.77192814853833 3458D + 04
x2670	0.190871096344010433D + 11	0.18384706080318 4605D + 03
sr16000	0.190871111864473018Q + 11	0.19087112111496 2521Q + 11

(**注**)SR16000**では他に変化の小さい要素がある。**

となっています。

あとSR16000でのあるλの範囲でのa_{-2}, a_{-1}, a_0は以下の様になっています。

ramda=10^{-16}~10~[-30}
a2= 4.000003447246776458440340467574 16
a1= 3.998614076193875546687457655832 65
a0= -12.263945340240600000000000000 20020

ramda=10^{-31}~10~[-45}
a2= 4.000000995778690564344849305288 36
a1= 3.999395092335383959734798438551 65
a0= -12.326990512798999999999999975 9162

ramda=10^{-46}~10~[-60}

a2= 4.000000412642012798006451400647 05
a1= 3.999663959451037435790618527291 21
a0= -12.358196281822999999999999967 96046

ramda=10^{-61}~10~[-75}

a2= 4.000000196766259218742464194527 51
a1= 3.999794900923913886687933816573 74
a0= -12.378147480874000000000000184813 6

ramda=10^{-76}~10~[-90}

a2= 4.000000144598692200929075901625 62
a1= 3.999832409009722650036422490660 11
a0= -12.384869002304000000000000160380 0

rambda=10^{-91}~10^{-105}

a2= 4.000000036947829850173391166644 84
a1= 3.999964948924225079755979080362 44
a0= -12.415458713321752600000000000000 01

λの値が小さくなるほど

a_0は$-\frac{5\pi^2}{3}+4$に近付いています。

3.4.3 bsgamma

bsγ planar

$$D = c(x_1(p_1^2 - m_1^2) + x_2(p_2^2 - m_2^2) - x_3 m_3^2 - x_6 m_6^2)$$
$$- c_1(x_5^2 p_2^2 + x_4^2 p_1^2 + x_4 x_5(p_1^2 + p_2^2))$$
$$- c_2(x_2^2 p_2^2 + x_1^2 p_1^2 + x_1 x_2(p_1^2 + p_2^2))$$
$$- 2x_2 x_3 x_5 p_2^2 - 2x_1 x_3 x_4 p_1^2 - x_3(x_2 x_4 + x_1 x_5)(p_1^2 + p_2^2)$$

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

$$c_1 = x_1 + x_2 + x_3$$

$$c_2 = 1 - x_1 - x_2$$

$$c = x_3(1 - x_1 - x_2 - x_3) + (x_1 + x_2)(1 - x_1 - x_2) = x_3(c_2 - x_3) + (1 - c_2)c_2$$

$$p_1^2 = m_b^2, p_2^2 = m_s^2, m_1^2 = m_2^2 = 1.5^2, m_3^2 = 80.3477^2,$$

$$m_b^2 = 4.7^2, m_s^2 = 0.094^2, m_6^2 = \lambda^2$$

$$I = \int_0^1 \int_0^{1-x_1} \int_0^{1-x_1-x_2} \int_0^{1-x_1-x_2-x_3} \int_0^{1-x_1-x_2-x_3-x_4} \frac{1}{D^2} dx_5 dx_4 dx_3 dx_2 dx_1$$

bsγ planar

5次元用の変数変換

$$x_1 = xy(1-z) \qquad x_1 + x_2 = xy$$
$$x_2 = xyz \qquad x_1 + x_2 + x_3 = x$$
$$x_3 = x(1-y) \qquad c_1 = x$$
$$x_4 = (1-x)u(1-v) \qquad c_2 = 1-xy$$
$$x_5 = (1-x)uv \qquad x_4 + x_5 = (1-x)u$$
$$x_6 = (1-x)(1-u) \qquad c = x[(1-x)(1-y) + y(1-xy)]$$
$$J = x^2(1-x)^2 yu$$
$$\frac{J}{D^2} = \frac{J_0}{D_0{}^2} \qquad J_0 = (1-x)^2 yu$$

$$\begin{aligned} D_0 = &-[(1-x)(1-y) + y(1-xy)][xym_1{}^2 + x(1-y)m_3{}^2 \\ &+ (1-x)(1-u)m_6{}^2] \\ &- (1-x)^2 u^2[(1-v)p_1{}^2 + vp_2{}^2] \\ &+ x(1-x)y(1-y)[(1-z)p_1{}^2 + zp_2{}^2 \\ &\quad - 2u(zvp_2{}^2 + (1-z)(1-v)p_1{}^2) \\ &\quad - u(z(1-v) + (1-z)v)(p_1{}^2 + p_2{}^2)] \end{aligned}$$

これを更に4次元化する。

4次元化後のbsγ

$$I = \int_{\Omega^5} \frac{(1-x)^2 yu}{D^2} d\Omega^5 = \int_{\Omega^4} \frac{(1-x)^2 yu}{(A+B)B} d\Omega^4$$

$$\begin{aligned}
D = {} & [1-x+xy(1-y)][xy(1-z)(p_1^2 - m_1^2) + xyz(p_2^2 - m_2^2)] \\
& -[1-x+xy(1-y)][x(1-y)m_3^2 + (1-x)(1-u)\lambda^2] \\
& -(1-x)^2 u^2 [(1-v)p_1^2 + vp_2^2] \\
& -(1-xy)xy^2 [(1-z)p_1^2 + zp_2^2] \\
& -x(1-x)y(1-y)u[(2-z-v)p_1^2 + (z+v)p_2^2]
\end{aligned}$$

$$A = (p_1^2 - p_2^2)(1-x)u[(1-x)u + xy(1-y)]$$

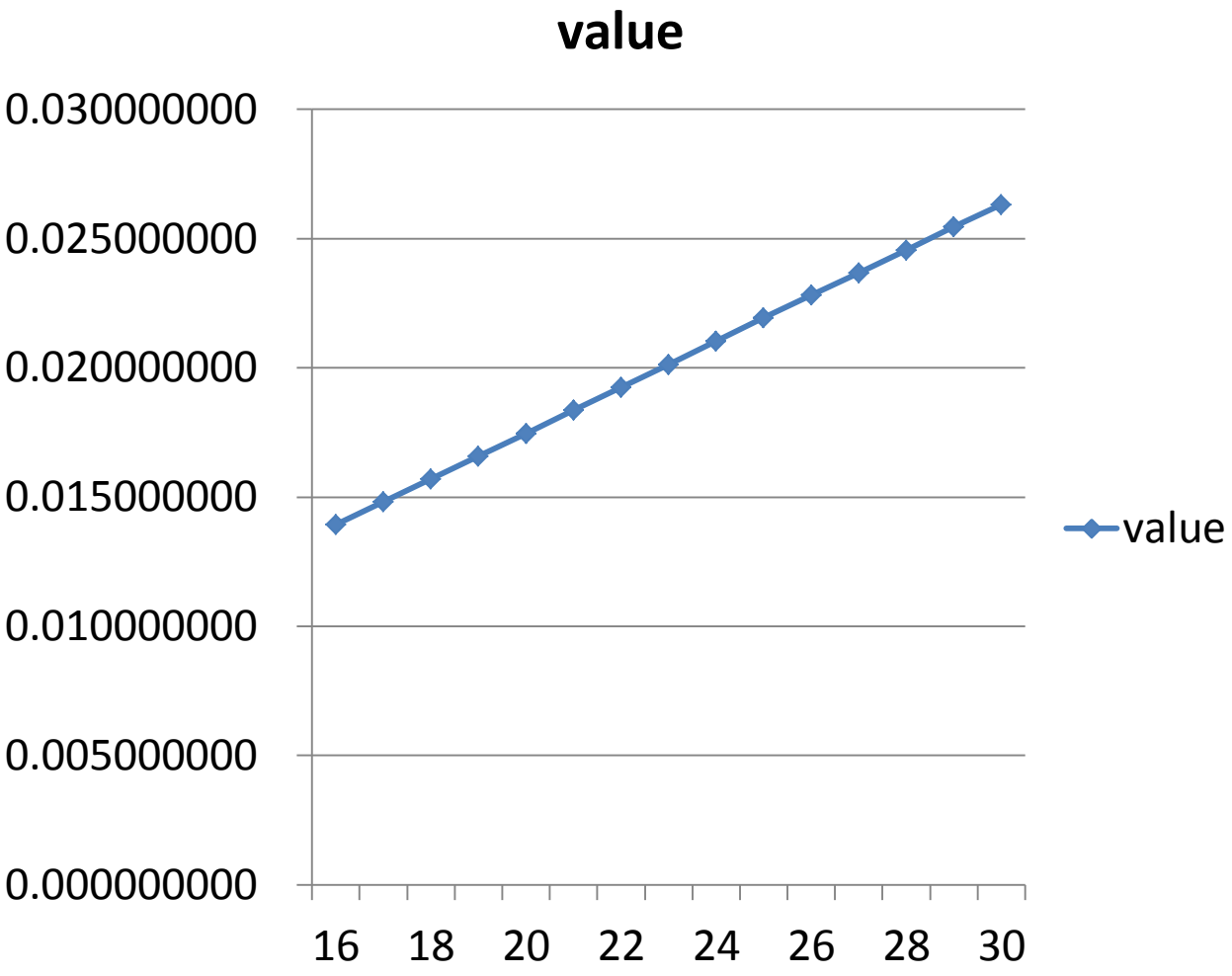
$$\begin{aligned}
B = {} & [1-x+xy(1-y)][xy(1-z)(p_1^2 - m_1^2) + xyz(p_2^2 - m_2^2)] \\
& -[1-x+xy(1-y)][x(1-y)m_3^2 + (1-x)(1-u)\lambda^2] \\
& -(1-x)^2 u^2 p_1^2 - x(1-x)y(1-y)u[(2-z)p_1^2 + zp_2^2] \\
& -(1-xy)xy^2 [(1-z)p_1^2 + zp_2^2]
\end{aligned}$$

$$\begin{aligned}
& p_1^2 = m_b^2, p_2^2 = m_s^2, m_1^2 = m_2^2 = 1.5^2, m_3^2 = 80.3477^2, \\
& \quad m_b^2 = 4.7^2, m_s^2 = 0.094^2, m_6^2 = \lambda^2
\end{aligned}$$

bsgamma planar

$\lambda = 10^{-n}$

n	value
16	0.013945911
17	0.014826698
18	0.015707767
19	0.016590158
20	0.017473234
21	0.018355518
22	0.019238562
23	0.020130180
24	0.021035437
25	0.021938463
26	0.022811082
27	0.023666177
28	0.024552533
29	0.025458382
30	0.026327504


value
0.030000000
0.025000000
0.020000000
0.015000000
0.010000000
0.005000000
0.000000000
16
18
20
22
24
26
28
30
value

$$\lambda = 10^{-n}$$

bsgamma 一次差分結果

n	N=728	N=1024	N=1456
16	0.000881676036303	0.000881676024318	0.000881676024308
17	0.000881675957819	0.000881676024256	0.000881676024308
18	0.000881676127056	0.000881676024379	0.000881676024308
19	0.000881676192593	0.000881676024096	0.000881676024308
20	0.000881675929067	0.000881676025850	0.000881676024307
21	0.000881675933258	0.000881676026869	0.000881676024308
22	0.000881675979610	0.000881676024874	0.000881676024305

3.4.5 two loop vertex

bsγ planar

を変形して, $p_1^2 = p_2^2 = m_1^2 = m_2^2 = m_\mu^2$ **として**,

g2**を作成する。**

$$D = -C(x_3 m_3^2 + x_6 m_6^2)$$
$$-[(x_1 + x_2 + x_3)(x_4 + x_5)^2 + (x_1 + x_2)^2(1 - x_1 - x_2)$$
$$+ 2x_3(x_1 + x_2)(x_4 + x_5)]m_\mu^2$$

807 $m_\mu = 0.1057, m_3 = m_6 = \lambda$

808 $m_\mu = 0.1057, m_3 = M_z = 91.19, m_6 = \lambda$

809 $m_\mu = 0.1057, m_3 = M_{\mathbf{H}} = 125.5, m_6 = \lambda$

変数変換

$x_1 = xy(1-z)$ $\quad x_1 + x_2 = xy$

$x_2 = xyz$ $\quad x_1 + x_2 + x_3 = x$

$x_3 = x(1-y)$ $\quad c_1 = x$

$x_4 = (1-x)u(1-v)$ $\quad c_2 = 1 - xy$

$x_5 = (1-x)uv$ $\quad x_4 + x_5 = (1-x)u$

$x_6 = (1-x)(1-u)$ $\quad c = x[(1-x)(1-y) + y(1-xy)]$

$J = x^2(1-x)^2 yu$

$\frac{J}{D^2} = \frac{J_0}{D_0^2}$ $\qquad J_0 = (1-x)^2 yu$

g2

$$D_0 = -[(1-x)(1-y) + y(1-xy)][x(1-y)m_3^2 + (1-x)(1-u)m_6^2]$$
$$-[xy^2(1-xy) + (1-x)^2 u^2 + 2x(1-x)y(1-y)u]m_\mu^2$$

と3**次元化して** $m_\mu^2 = 1$ **と正規化して計算する。**

15**の** $\lambda = 10^{-n}(16 \le n \le 30)$ **に対して計算し**,807**は二次差分を**,808,809

は一次差分をとり, $\log(\frac{1}{\lambda^2})$ **の二次式**,**一次式となる事を検証。**

g2　808一次差分
N=2912で10進11桁（全桁）一致

	g2 808 一次差分		
n	N=728	N=1456	N=2912
16	0.000038733241268	0.000038733248507	0.000038733248507
17	0.000038733234064	0.000038733248507	0.000038733248507
18	0.000038733266331	0.000038733248507	0.000038733248507
19	0.000038733326864	0.000038733248507	0.000038733248507
20	0.000038733174086	0.000038733248506	0.000038733248507
21	0.000038733353236	0.000038733248506	0.000038733248507
22	0.000038733544343	0.000038733248510	0.000038733248507
23	0.000038731850666	0.000038733248499	0.000038733248507
24	0.000038732625311	0.000038733248540	0.000038733248507
25	0.000038734471892	0.000038733248546	0.000038733248507
26	0.000038731896322	0.000038733248424	0.000038733248507
27	0.000038724804488	0.000038733248425	0.000038733248507
28	0.000038738845469	0.000038733249112	0.000038733248507
29	0.000038740941884	0.000038733249063	0.000038733248507

g2　809一次差分
N=2912で10進11桁（全桁）一致

	g2 809一次差分		
n	N=728	N=1456	N=2912
16	0.000021493099355	0.000021493102821	0.000021493102821
17	0.000021493093286	0.000021493102821	0.000021493102821
18	0.000021493115638	0.000021493102821	0.000021493102821
19	0.000021493142483	0.000021493102821	0.000021493102821
20	0.000021493082925	0.000021493102821	0.000021493102821
21	0.000021493117428	0.000021493102821	0.000021493102821
22	0.000021493302447	0.000021493102823	0.000021493102821
23	0.000021492401183	0.000021493102817	0.000021493102821
24	0.000021492629664	0.000021493102839	0.000021493102821
25	0.000021493704654	0.000021493102844	0.000021493102821
26	0.000021492781172	0.000021493102769	0.000021493102821
27	0.000021488800060	0.000021493102792	0.000021493102821
28	0.000021495565878	0.000021493103126	0.000021493102821
29	0.000021496356431	0.000021493103198	0.000021493102821

3.4.4 4次元積分

4次元積分として,以下のS^{221}を検討した。

$$S^{221}(s;m_1^2,m_2^2,m_3^2,m_4^2,m_5^2)=\int_0^1\int_0^{1-x}\int_0^{1-x-y}\int_0^{1-x-y-z}\frac{1}{DC}dudzdydx$$

$$C=(x+y+z+u)(1-x-y-z-u)+(x+y)(z+u)$$

$$E=(1-x-y-z-u)(x+z)(y+u)+(x+y)zu+(z+u)xy$$

$$M^2=xm_1^2+ym_2^2+zm_3^2+um_4^2+(1-x-y-z-u)m_5^2$$

$$D=-sE+M^2C$$

積分領域はΩで表す。

精度チェックのための 基本演算の解析解と計 算結果は下記のとうり。

解析解

case1 $S^{221}(1;0,0,0,0,0)=-6\varsigma(3)=-7.21234141895756568$

case2 $S^{221}(-1;0,0,0,0,0)=6\varsigma(3)=7.21234141895756568$

case3 $S^{221}(0;1,1,1,1,0)=1$

case4 $S^{221}(0;0,1,1,1,0)=\varsigma(2)=\frac{\pi^2}{6}=1.644934066848226436472$

case5 $S^{221}(0;1,1,1,1,1)=-\int_0^1\frac{x(\frac{\ln x(1-x)}{1-x+x^2}+1)}{1-x+x^2}=0.78130241289648$

計算解

case1 result = - 7.21234141895747793

case2 result = 7.21234141895747793

case3 result = 0.999999999999997335

case4 result = 1.64493406684822352

case5 result = 0.781302412896484055

特殊な計算例は以下のとうり

$S^{221}(s;m^2,0,0,0,0)$　　$s=\pm 1, m^2=1$　　**解析解** $\mp 3\zeta(3)=\mp 3.6061707947878284$

$s=-1, m^2=1$

計算解(ζ - 算法なし)= 3.60617070947862928

計算解(y = -s/m^2) = 0.107304279342169218 + 162　0.363 + 149

計算解(x = -m^2/s) = 0.36061706993083602ID + 01　0.257D - 10

$s=1, m^2=1$

計算解(ζ - 算法なし)= + **********

計算解(y = -s/m^2) = - 0.2002135660258784 17 + 162　0.261 + 162

計算解(x = -m^2/s) = - 0.3606118400861984 87D + 01　0.161D - 04

尚以下の式は発散する。

$$S^{221}(0;0,0,1,1,1,0)=-\int_0^1 \frac{\ln(1-x)}{x^2}dx$$

$$S^{221}(0;0,0,0,1,1,0)=-\int_0^1 \frac{\ln(1-x)}{x^2}dx\times\int_0^1 \frac{1}{y}dy$$

さらに詳細に解析すると

$$S^{221}(s;m^2,0,0,0,0) = \int_0^1 \int_0^{1-x} \int_0^{1-x-y} \int_0^{1-x-y-z} \frac{1}{CD} dudzdydx = \int_\Omega \frac{1}{CD} d\Omega$$

$$D = -sE + xm^2C \qquad s = -1, m^2 = 1 \quad => D = E + xC$$

$$\frac{1}{CD} = \frac{1}{CE} - \frac{x}{E(E+xC)}$$

$$\int_\Omega \frac{1}{CE} d\Omega = 6\varsigma(3) \quad => \textbf{実測結果} \quad \text{result} = \ 7.21234141895747793$$

$$\int_\Omega \frac{x}{E(E+xC)} d\Omega = 3\varsigma(3) \quad => \textbf{実測結果} \quad \text{result} = \ 3.60617070947877494$$

$$\int_\Omega \frac{1}{C^2} d\Omega = S^{221}(0;1,1,1,1,1) \quad => \textbf{実測結果} \quad \text{result} = \ 0.781302412896484055$$

$$\int_\Omega \frac{1}{C} d\Omega = \frac{1}{6} \quad => \textbf{実測結果} \quad \text{result} = \ 0.16666666666666665747$$

$$\int_\Omega \frac{1}{E} d\Omega = \frac{11}{9}\varsigma(3) \quad => \textbf{実測結果} \quad \text{result} = \ 1.46918065941728160$$

$S^{221}(-1;1,1,1,1,1)$ result = 0.680876150423024185

$S^{221}(1;1,1,1,1,1)$ result = 0.923631826519863641

尚, 以下の場合は発散する。

$$\int_\Omega \frac{1}{xC^2} d\Omega = -\int_0^1 \frac{\ln(1-x)}{1-x} \ dx$$

$m^2 = 50$と固定し，sを5,10,15,...,100と変化させたときの二重指数関数型積分の積分結果は以下のとうり.二重指数関数型積分のサイズは$n = 1024, \varepsilon -$算法でのεの初期値は1.2^{-15}

s	積分値	積分誤差
5	0.142933799933564448D+00	0.864D-06
10	0.874770761633400645D-01	0.495D-04
15	0.580754664645131494D-01	0.123D-04
20	0.377928730070589761D-01	0.444D-06
25	0.218394449541923323D-01	0.113D-03
30	0.773426500218699423D-02	0.511D-05
35	-0.570584090394023972D-02	0.168D-04
40	-0.198070245154403426D-01	0.125D-06
45	-0.369777391702599822D-01	0.383D-04
50	-0.721357975080699487D-01	0.333D-05
55	-0.989714303728692485D-01	0.183D-05
60	-0.102840960786493671D+00	0.923D-06
65	-0.101914671715339883D+00	0.185D-06
70	-0.991426158779158406D-01	0.275D-05
75	-0.956234990743119462D-01	0.330D-06
80	-0.917437470939333061D-01	0.166D-03
85	-0.877601377899803609D-01	0.324D-03
90	-0.848383517809659093D-01	0.422D-03
95	-0.814307958141493454D-01	0.575D-03
100	-0.772582671356500383D-01	0.244D-04

さらに複雑にした場合の

$S^{221}(s;m^2,m^2,0,0,m^2)$　**計算**

$m^2=100$**と固定し**,s**を**5,10,15,...,100**と変化させたときの二重指数関数型積分の積分結果は以下のとうり.二重指数関数型積分のサイズは**$n=1024,\varepsilon-$**算法での**ε**の初期値は**1.2^{-15}

s	積分値	積分誤差
5	0.300712966040258094D-01	0.109D-06
10	0.266732292799612622D-01	0.190D-08
15	0.247016613851884191D-01	0.650D-07
20	0.233122764814605241D-01	0.675D-08
25	0.222407169211559473D-01	0.115D-08
30	0.213693658017195323D-01	0.797D-08
35	0.206356969069118670D-01	0.595D-06
40	0.200022139331064649D-01	0.213D-08
45	0.194450748750508856D-01	0.746D-08
50	0.189476908636132708D-01	0.297D-06
55	0.184989268813915836D-01	0.590D-07
60	0.180892912432545142D-01	0.141D-06
65	0.177130501245448997D-01	0.180D-07
70	0.173657808967870794D-01	0.151D-06
75	0.170416613603775538D-01	0.238D-06
80	0.167388840687285462D-01	0.178D-06
85	0.164538254991338749D-01	0.124D-07
90	0.161835797358507970D-01	0.157D-05
95	0.159310205462771873D-01	0.299D-06
100	0.156908197395704066D-01	0.802D-06

3.4.6 3loop積分

今回扱った3loop**積分は**

$$\int_0^1 \int_0^{1-x_1} \int_0^{1-x_1-x_2} \int_0^{1-x_1-x_2-x_3} \int_0^{1-x_1-x_2-x_3-x_4} \int_0^{1-x_1-x_2-x_3-x_4-x_5} \int_0^{1-x_1-x_2-x_3-x_4-x_5-x_6} \frac{1}{D^2} d\Omega$$

$$d\Omega = dx_7 dx_6 dx_5 dx_4 dx_3 dx_2 dx_1$$

でDが次の**3ケース**$(N0, L01, L02)$**の場合です。**

解析解は$20\zeta_5 = 20.7385551028673985...$**。**

NO

x8=1.0-x1-x2-x3-x4-x5-x6-x7

x23=x2+x3

x24=x2+x4

x247=x2+x4+x7

x1235=x1+x2+x3+x5

x2346=x2+x3+x4+x6

x5678=x5+x6+x7+x8

D = x24**2*x5**2-x2346*x247*x5**2-x2**2*x2346*x5678
+2*x2*x23*x24*x5678-x1235*x24**2*x5678
-x23**2*x247*x5678+x1235*x2346*x247*x5678
+2*x2*x24*x5*x6-2*x23*x247*x5*x6+x2**2*x6**2
-x1235*x247*x6**2+2*x2*x2346*x5*x7
-2*x23*x24*x5*x7+2*x2*x23*x6*x7
-2*x1235*x24*x6*x7+x23**2*x7**2-x1235*x2346*x7**2

L01

x8=1.0-x1-x2-x3-x4-x5-x6-x7
x123=x1+x2+x3
x167=x1+x6+x7
x348=x3+x8+x4
x2578=x2+x5+x7+x8

D = -x167*x2578*x3**2-x167*x2**2*x348-x1**2*x2578*x348
+x123*x167*x2578*x348-2*x1*x2*x348*x7+x3**2*x7**2
-x123*x348*x7**2-2*x167*x2*x3*x8-2*x1*x3*x7*x8
+x1**2*x8**2-x123*x167*x8**2

L02

x8=1.0-x1-x2-x3-x4-x5-x6-x7
x123=x1+x2+x3
x1456=x1+x4+x5+x6
x678=x6+x7+x8
x247=x2+x4+x7

D = x2**2*x6**2-x123*x247*x6**2-x1456*x2**2*x678
-x1**2*x247*x678+x123*x1456*x247*x678-2*x1*x2*x4*x678
-x123*x4**2*x678-2*x1*x2*x6*x7-2*x123*x4*x6*x7
+x1**2*x7**2-x123*x1456*x7**2

L02での計算手順例

(1) h=0.2 -2.4<t<2.4

とてつもない結果

3.4834220858009O6E-008 0.999999965165779

n= 25
h= 0.200000000000000
result= 1463432.72964967

(2) hの目安をつける。
10進5桁まで一致
h=4.35/24

1.182374362483700E-006 0.999998817625638

n= 25
h= 0.181250000000000
result= 20.7383465515643

(3)分点数を増やす
　効果なし.

1.182374362483700E-006 0.999998817625638
　n=　　41
h= 0.108750000000000
result=　20.7294941799961

(4)解析値を超すhを求める。
　　h=4.36/24

1.102223275929152E-006 0.999998897776724

　n=　　25
h= 0.181666666666667
result=　20.7866519155393

(5) h=(4.351から4.359)/24で計算
　結果が解析値のまわりを振動する。

演算精度を上げて（1）から（5）まで行い
振動しなければ二分法により精度を上げていく
ことが出来る。

3.4.7 4loop積分

今回扱った4loop**積分は**

$$\int_0^1 \int_0^{1-x_1} \int_0^{1-x_1-x_2} \int_0^{1-x_1-x_2-x_3} \int_0^{1-x_1-x_2-x_3-x_4} \int_0^{1-x_1-x_2-x_3-x_4-x_5} \int_0^{1-x_1-x_2-x_3-x_4-x_5-x_6} \int_0^{1-x_1-x_2-x_3-x_4-x_5-x_6-x_7} \frac{1}{CD} d\Omega = dx_8 dx_7 dx_6 dx_5 dx_4 dx_3 dx_2 dx_1$$

でC, D**が次の**2**ケース**(M_{44}, M_{45})**の場合です。**

解析解は

M_{44}**は** $\frac{441\zeta_7}{8} = 55.5852539156784...$**。**

M_{45}**は** $36\zeta_3{}^2 = 52.017868743610.....$**。**

M_{44}

```
x9 = 1.0 - x1 - x2 - x3 - x4 - x5 - x6 - x7 - x8
     x123 = x1 + x2 + x3
     x167 = x1 + x6 + x7
     x278 = x2 + x7 + x8
     x349 = x3 + x4 + x9
     x589 = x5 + x8 + x9
     C = x167 * x278 * x349 * x589 - x349 * x589 * x7 **2 - x167 * x349 * x8 **2
      - x167 * x278 * x9 **2 + x7 **2 * x9 **2
     D = - (x167 * x278 * x3 **2 * x589) - x167 * x2 **2 * x349 * x589
       - x1 **2 * x278 * x349 * x589
       + x123 * x167 * x278 * x349 * x589 - 2 * x1 * x2 * x349 * x589 * x7
       + x3 **2 * x589 * x7 **2
       - x123 * x349 * x589 * x7 **2 + x167 * x3 **2 * x8 **2
       + x1 **2 * x349 * x8 **2
       - x123 * x167 * x349 * x8 **2 - 2 * x167 * x2 * x3 * x8 * x9
       - 2 * x1 * x3 * x7 * x8 * x9
       + x167 * x2 **2 * x9 **2 + x1 **2 * x278 * x9 **2
       - x123 * x167 * x278 * x9 **2
       + 2 * x1 * x2 * x7 * x9 **2 + x123 * x7 **2 * x9 **2
```

M_{45}

```
x9 = 1.0 - x1 - x2 - x3 - x4 - x5 - x6 - x7 - x8
    x123 = x1 + x2 + x3
    x1567 = x1 + x5 + x6 + x7
    x279 = x2 + x7 + x9
    x349 = x3 + x4 + x9
    x589 = x5 + x8 + x9
    c = -x279 * x349 * x5 * *2 + x1567 * x279 * x349 * x589 - x349 * x589 * x7 * *2
     - 2 * x349 * x5 * x7 * x9 - x1567 * x279 * x9 * *2 - x1567 * x349 * x9 * *2
     + x5 * *2 * x9 * *2 - x1567 * x589 * x9 * *2 + 2 * x5 * x7 * x9 * *2
     + x7 * *2 * x9 * *2 + 2 * x1567 * x9 * *3
    d = x279 * x3 * *2 * x5 * *2 + x2 * *2 * x349 * x5 * *2 - x123 * x279 * x349 * x5 * *2
       - x1567 * x279 * x3 * *2 * x589 - x1567 * x2 * *2 * x349 * x589
       - x1 * *2 * x279 * x349 * x589 + x123 * x1567 * x279 * x349 * x589
       - 2 * x1 * x2 * x349 * x589 * x7 + x3 * *2 * x589 * x7 * *2
       - x123 * x349 * x589 * x7 * *2 + 2 * x1 * x279 * x3 * x5 * x9
    - 2 * x1 * x2 * x349 * x5 * x9 + 2 * x2 * x3 * x5 * *2 * x9 - 2 * x1567 *
    x2 * x3 * x589 * x9
    + 2 * x2 * x3 * x5 * x7 * x9 + 2 * x3 * *2 * x5 * x7 * x9 - 2 * x123 * x349 *
    x5 * x7 * x9
    - 2 * x1 * x3 * x589 * x7 * x9 + x1567 * x2 * *2 * x9 * *2 + x1 * *2 * x279 * x9 * *2
    - x123 * x1567 * x279 * x9 * *2 + 2 * x1567 * x2 * x3 * x9 * *2
    + x1567 * x3 * *2 * x9 * *2 + x1 * *2 * x349 * x9 * *2 - x123 * x1567 * x349 * x9 * *2
    + 2 * x1 * x2 * x5 * x9 * *2 - 2 * x1 * x3 * x5 * x9 * *2 + x123 * x5 * *2 * x9 * *2
    + x1 * *2 * x589 * x9 * *2 - x123 * x1567 * x589 * x9 * *2 + 2 * x1 * x2 * x7 * x9 * *2
    + 2 * x1 * x3 * x7 * x9 * *2 + 2 * x123 * x5 * x7 * x9 * *2 + x123 * x7 * *2 * x9 * *2
    - 2 * x1 * *2 * x9 * *3 + 2 * x123 * x1567 * x9 * *3
```

M_{44}計算作業手順

(1)DEの積分変数変換区間$[\varepsilon, 1-\varepsilon]$でそれぞれの演算精度に応じて$\varepsilon = 2^{-53}$(倍精度), $\varepsilon = 2^{-65}$(拡張倍精度), $\varepsilon = 2^{-113}$(4倍精度)で計算。全てnanまたはゼロ割りが発生。

(2)$\varepsilon = 2^{-53}$(4倍精度)$(h = 0.25)$, 倍精度$t = -2.4$から$t = 2.4$, 拡張倍精度$t = -2.6$から$t = 2.6(h = 0.2)$にして実行.
結果の精度は倍精 度演算, 拡張倍精度, 4倍精度の順.
すなわち計算結 果精度はεではなくhに大きく依存する.

(3)$t = -2.6$から$t = 2.6$倍精度演算で実行すると, nanが発生。
これは$\varepsilon^2 > 2^{-53}$の条件を満たしていない。

(4)$\varepsilon = 2^{-m+1}$ (m = 仮数部のビット数, $l = \dfrac{\text{乗算の最高次数} - 1}{2} = 2$)
より, $\varepsilon = 2^{-50}$, tの絶対値最大値t_{max}

$$t_{max} = \sinh^{-1}\frac{\log(\frac{1}{\varepsilon})}{2\pi}, h = t_{max}/(N-1) \qquad N = 2n+1$$

$$t_{max} = 2.408897648$$

(5)$\varsigma(7) = \dfrac{1}{\Gamma(7)}\displaystyle\int_0^\infty \frac{t^6}{e^t - 1}dt$より

$$\int_0^\infty \frac{t^6}{e^t - 1}dt = \int_0^1 \frac{(\log(x))^6}{1-x}dx$$

からDEの項が最大になるのは, $t = -1.5$付近。これにより, Nを固定しhをわずかずつ増やす方法を取る事にした。

(6)t = 2.4からt = 2.44までの最適値付近ではtの絶対値の単調増加の値を示すので,(Nを固定して)一桁ずつ計算。一回の 計算で 200×9秒 = 30分で5回実行すれば,求める値にたどりつく.
(1回で一桁精度が向上し, t = −2.4で10進5桁の精度があったので。)

E5 – 2670, Phi5110Pでの実行結果はすべて10進11桁一致

E5 – 2670　FORTRAN　倍精度,4倍精度
　　C　倍精度 , 拡張倍精度
Phi5110P　FORTRAN　倍精度
　　C　倍精度 , 拡張倍精度
最適化オプションは倍 精度, 拡張倍精度
　– O1　– openmp　(−mmic)
4倍精度は – O1　– openmp,−O2　– openmp

M_{45}計算作業手順

(1)tのとる範囲をM_{44}の結果をもとに$t = -2.4$から$t = 2.4$までと想定した。

(2)3loopの3ケースとM_{45}では倍精度演算では$\mathrm{nan}, \mathrm{inf\ inity}$が発生したため,拡張倍精度,DD形式4倍精度演算を使用。

(3)時間を考え,3loopはSR16000で4倍精度で実行。

(4)M_{45}はE5 – 2670　拡張倍精度で実行し,最終結果をSR16000で4倍精度で実行。

(5)M_{45}の結果は10進10桁一致,11桁一致,12桁一致する結果が得られた。(次ページ)

(6)M_{45} 4倍精度はSR16000で実行中。
10進8桁一致する場合の実行 時間は6374秒。

(7)オプション　E5 – 2670　– O1　– openmp　(16smp)
SR16000　– O3　– omp　(64smp)
サイズは全て$N = 25$

拡張倍精度実行結果

E5 - 267016smp
拡張倍精度　　N ＝ 25

h ＝ 4.785582/24　　10桁一致
x30[0] ＝ 3.9488418198428373857 0e - 08
x30[n - 1] ＝ 9.99999960511581801627e - 01
size ＝　　25
h ＝ 1.9939925000000000e - 01
eps ＝ 0.0000000000000000e ＋ 00
result ＝ 5.2017868748703023 2e ＋ 01
wallclock time ＝ 688.694365 sec

h = 4.7855815/24　11桁一致
x30[0] = 3.9488589324982980923 7e - 08
x30[n - 1] = 9.99999960511410675059e - 01
size =　　25
h = 1.9939922916666666 4e - 01
eps = 0.0000000000000000e + 00
result = 5.2017868743216005 3e + 01
wallclock time = 688.619698 sec

h = 4.78558154/24　12桁一致
x30[0] = 3.9488575634832879260 3e - 08
x30[n - 1] = 9.99999960511424365172e - 01
size =　　25
h = 1.9939923083333333 0e - 01
eps = 0.0000000000000000e + 00
result = 5.2017868743634485 2e + 01
wallclock time = 688.669904 sec

3.4.8　3loop,4loop積分作業手順まとめ

DEでの変数変換区間$[\varepsilon,1-\varepsilon]$として,それに対応する
tの値をそれぞれ$-t_{max},t_{max}(t_{max}>0)$とする。

$$t_{max}=\sinh^{-1}(\frac{\log\frac{1}{\varepsilon}}{\pi})$$

目安としては$\varepsilon \fallingdotseq 10^{-8}$は$t_{max}=2.4,\varepsilon \fallingdotseq 10^{-14}$は$t_{max}=3.0$
倍精度演算で$t_{max}=2.4$でまともな値が出たのは,M_{44}のみ.
$M_{45},N0,L0(case1),L0(case2)$ではnan,inf inity,$10^6$という結果。
ここでt_{max}の値を小さくしてある程度妥当な値がでた場合
のt_{max}の最大値は$L0(case2)$の$t_{max}=2.175$.
ここで,$L0(case2)$で$t_{max}=2.175$を少しずつ変えてより良い
精度の結果を求めようとすると,値が解析値のまわりを振動
する.M_{44}の場合は解析値付近では単調増加になっていて
より良い精度が求められた。
$L0(case2)$の振動の原因を調べるため,

$$\Gamma(5)\zeta(5)=\int_0^1\frac{(\log(x))^4}{1-x}dx$$

で調査した。(この式だと倍精度演算
でも,$t_{max}=3.0$でも計算可能なため)
$t_{max}=2.175$の場合

```
i,f =        1  0.574525277298576
i,f =        2   2.338902199740 41
i,f =        3   6.506187034526 39
i,f =        4   13.13858011640 68
i,f =        5   20.21485378889 24
i,f =        6   24.58895788821 03
i,f =        7   24.26987044094 96
i,f =        8   19.74338277379 06
i,f =        9   13.31480781532 39
i,f =       10   7.424437100065 66
i,f =       11   3.400833478140 04
i,f =       12   1.327582059574 75
i,f =       13  0.3625949249496 45
```

ここで,iとfは,
$t=(-13+i)\times h$,と$(26-i)\times h$の場合の
被積分関数×重み係数の和。
$i=13$は$t=0$の場合の被積分関数×重み係数。
ここでわかるとうり,$i=13$の場合が最小値となって
いて,$i=1$の場合の方が大きい.これは,DEの持つ,
tの絶対値が大きいところでは,急速に値が減少
すると言う性質を表わしていない。t_{max}を変更せず
$N=25$を$N=41$に変えてもL0(case2)の精度は改善
されなかった事や,t_{max}を少しかえても値が振動する
のはこれが原因による.

$t_{max}=2.4$だと$i,f=$ 1 5.000558641136945E - 002

$t_{max}=3$だと $i,f=$ 1 6.662369942894555E - 007

$i,f=$ 2 2.085441754992190E - 004

$i,f=$ 3 1.398608912042730E - 002

result = 24.8862661234180

kai = 24.8862661234409

$i=13$の値はt_{max}の値(Nを固定した場合のh)には依存しない。

$\frac{\pi}{2}(\log(2))^4$

すなわち,t_{max}の値を少しずつ変えて精度の良い
結果を得るには,$t=-t_{max},t_{max}$の場合の
被積分関数×重み係数の和が十分小さいかどうか
($t=0$での値は一定なのでこの時の値に対して)
が判断基準となります。

以上の結果から次の手順が考えられます。

(1)倍精度演算で$\varepsilon = 10^{-8}$に対する,$t_{max} = 2.4, N = 25$
,$h = 0.2$で妥当と思われる値が出るかどうか？
($\varepsilon = 10^{-8}$は$\varepsilon^2 = 10^{-16}$から来ている。)

(2)妥当と思われる値が出てれば,t_{max}のとき
被積分関数×重み係数の和が十分小さい
値かどうか？
この方法には以下の3つの方法が考えられる。

(ア)同じt_{max} で$N = 25$から$N = 41$で精度改善があるか
調べる.(DEの性質を満たしていれば,誤差1/4以下
となる.)
(演算量から言うと6次元積分$N = 128$に対し,
7次元積分$N = 64$,8次元積分 $= 38$で同じになり
倍精度の6次元積分の$N = 128$での実行時間
はわかっているため。)

(イ)全次元$t = 0$の場合の被積分関数×重み係数の値
と,一次元だけ,$-t_{max,}$とt_{max}のでの
被積分関数×重み係数の値の和を比較.
(n次元積分の場合,それぞれn個の値を比較
して,問題のある次元はないかチェック)

(ウ)全次元$t = 0$の場合の被積分関数×重み係数の値
と最小値(被積分関数×重み係数の値)及び
全次元$t = 0$の場合の被積分関数×重み係数の値×閾値
以下となる被積分関数×重み係数の個数でチェック
する.

(3)問題のある場合は拡張倍精度,4倍精度で,
$h = 0.2, N = 25, t_{max} = 2.4$から始める。可能ならば拡張倍精度
から始めるのが良い.
今回の実行では3loopは4倍精度演算,M_{45}は拡張倍精度
で結果が得られ,M_{45}のE5 − 2670(16smp)拡張倍精度演算
はSR16000/M1 (64smp)4倍精度演算の10倍(実行時間
1/10)となっている。

3.5　量子モンテカルロ法による物性スペクトル計算

量子モンテカルロ法では，安定した
計算を行うにはLは$L = 20 \times \beta$の関係が
満たされている事が必 要になります。
$U = 10$ではプログラムを実行 中に
変数がとる値の最大値 は
$e^{\sqrt{200} \times \beta}$となり，指数部が11ビットだと
$\beta < 50.14$,15ビットだと$\beta < 802.978$
となります。
また粒子数Nは100程度が必要となります 。
結果の精度からみるとNが20でも100とは
あまり差は出ていません。

3.5.1 指数部が11ビットの限界について

量子モンテカルロ法

$\beta =$ **可変**, $L = 20 \times \beta, U = 10$

倍精度の最大値 $1.7976931348623157E+0308$

```
l =         980
beta =   49.000000000000000000000000000000

ispin =           1
QDR  =  1.78586413917219442688347602650038OE+0303
I+QDR =  1.78586413917219442688347602650038OE+0303

ispin =           2
QDR  =  1.01692669526068116082611053984090 9E+0304
I+QDR =  1.01692669526068116082611053984090 9E+0304
l =        1000
beta =   50.000000000000000000000000000000

spin =           1
QDR  =  2.86271224279475689155735295986131 2E+0309
I+QDR =  2.86271224279475689155735295986131 2E+0309

ispin =           2
QDR  =  1.65160497057247654019491436843756 7E+0310
I+QDR =  1.65160497057247654019491436843756 7E+0310
```

$\beta = 49.6$ OK, $\beta = 49.7$ NG

```
l =        992
beta =   49.600000000000000000000000000000

ispin =          1
QDR  =  9.436165589130513054255944449201070E+0306
I+QDR =  9.436165589130513054255944449201070E+0306

ispin =          2
QDR  =  5.415677874136262454106581048120274E+0307
I+QDR =  5.415677874136262454106581048120274E+0307

l =        994
beta = 49.700000000000000000000000000000

ispin =          1
QDR  =  3.938127827940264707461183253762617E+0307
I+QDR =  3.938127827940264707461183253762617E+0307

ispin =          2
QDR  =  2.263160356072587757351389081936904E+0308
I+QDR =  2.263160356072587757351389081936904E+0308

l =        996
beta =   49.800000000000000000000000000000

ispin =          1
QDR  =  1.643557573940390154190429173363875E+0308
I+QDR =  1.643557573940390154190429173363875E+0308

ispin =          2
QDR  =  9.457544390246214860747340892604545E+0308
I+QDR =  9.457544390246214860747340892604545E+0308

l =        998
beta =   49.900000000000000000000000000000

ispin =          1
QDR  =  6.859317195871621637905579118618770E+0308
I+QDR =  6.859317195871621637905579118618770E+0308

ispin =          2
QDR  =  3.952228149418581143027425919256247E+0309
I+QDR =  3.952228149418581143027425919256247E+0309
```

3.5.2 必要演算精度見積り

βと必要演算精度は以下の様に見積もれます。

限界テスト

$\beta / L = a$(**一定**)$, U = 10$**とする。**$(0 < a << 1)$

$\beta = 10, L = 200, a = 0.05$

$$\sqrt{U\beta L} = \beta\sqrt{\frac{U}{a}} = 10\sqrt{2} \times \beta$$

必要ビット数 $= \ln_2 e^{\beta \times 10\sqrt{2}} = \dfrac{10\sqrt{2} \times \beta}{\ln 2} = 20.40278893 \times \beta$

β	必要ビット数	必要精度	L
100	2041	68	2000
180	3673	128	3600
250	5101	188	5000
300	6121	248	6000
400	8162	308	8000
500	10202	368	10000
600	12242	428	12000
700	14282	488	14080
800	16323	548	16000

548倍精度演算では限界に近い事もありβ＝800では他の演算精度の場合と異なる結果となる。

```
n=              8
l=          16000
beta, dt=    800.00000000000000000000000000000000
 5.00E-02
u, t0=    10.00000000000000000000000000000000
              1
fac=  0.70710678118654752440084436210484 9
For denominator: mstep=              1
For green func: mstep2=              1
ispin=              1
QDR  =  1.13949637284986456231601074493146 7E+4911
I+QDR=  1.13949637284986456231601074493146 7E+4911
ispin=              2
QDR  =  4.84493967547754402433739442571308 3E+4911
I+QDR=  4.84493967547754402433739442571308 3E+4911
   0.0000000000D+00    -0.2500000000D+00
```

```
n=              8
l=          15744
beta, dt=    800.00000000000000000000000000000000
 5.08E-02
u, t0=    10.00000000000000000000000000000000
              1
fac=  0.71283243564025129743160436923030 5
For denominator: mstep=              1
For green func: mstep2=              1
ispin=              1
QDR  =  3.37063287003268167588917330241850 4E+4929
I+QDR=  3.37063287003268167588917330241850 4E+4929
ispin=              2
QDR  =  1.05636585510384055575844764887041 4E+4931
I+QDR=  1.05636585510384055575844764887041 4E+4931
   0.0000000000D+00    -0.1000000000D+01
```

各演算精度でまとめた結果は下記のとうり。

調査一覧

項番	β	N	U	L	精度
1	100	100	10	2000	68倍精度
2	180	100	10	3600	128倍精度
3	250	100	10	5000	188倍精度
4	300	100	10	6000	248倍精度
5	400	20	10	8000	308倍精度
6	500	20	10	10016	368倍精度
7	600	20	10	12032	428倍精度
8	10000/14	20	10	14336	488倍精度
9	770	20	10	15424	548倍精度
10	780	20	10	15680	548倍精度
11	790	20	10	15808	548倍精度
12	800	20	10	15680	548倍精度
13	800	20	10	15744	548倍精度

最小値一覧

備考	理論値	実測値
1	0.567D-307	0.330D-307
2	0.120D-552	0.512D-553
3	0.130D-767	0.468D-768
4	0.370D-921	0.121D-921
5	0.299D-1228	0.927D-1229
6	0.143D-1536	0.424D-1537
7	0.686D-1845	0.199D-1845
8	0.296D-2197	0.849D-2198
9	0.176D-2366	0.705D-2367
10	0.246D-2401	0.703D-2402
11	0.159D-2426	0.454D-2427
12	0.625D-2432	0.179D-2432
13	0.688D-2437	0.197D-2437

論文リスト

外国雑誌

1.Acceleration of Feynman loop integrals in high-energy physics on many core GPUs,F. Yuasa, T. Ishikawa, N. Hamaguchi, T. Koike N. Nakasato. 2013. 7 pp. J.Phys.Conf.Ser. 454 (2013) 012081

2.Numerical Computation of Two-loop Box Diagrams with Masses, F. Yuasa , E. de Doncker, N. Hamaguchi, T. Ishikawa, K. Kato, Y. Kurihara, J. Fujimoto, Y. Shimizu . Dec 2011. 20 pp. Comput.Phys.Commun. 183 (2012) 2136-2144

国際会議プロシーディング

3.Numerical approach to multi-loop integrals K. Kato, E. de Doncker, N. Hamaguchi, T. Ishikawa, T. Koike, Y. Kurihara, Y. Shimizu, F. Yuasa. Jan 2012. 7 pp. PoS QFTHEP2011 (2013) 029

4. The GRACE project: QCD, SUSY, multi-loop J. Fujimoto et al.. 2011. 8 pp, PoS RADCOR2011 (2011) 012

5. Numerical Approach to Calculation of Feynman Loop Integrals F. Yuasa, T. Ishikawa, Y. Kurihara, J. Fujimoto, Y. Shimizu, N. Hamaguchi , E. de Doncker , K. Kato . Sep 2011. 8 pp. PoS CPP2010 (2010) 017

6. Recursive box and vertex integrations for the one-loop hexagon reduction in the physical region E. de Doncker, J. Fujimoto, N. Hamaguchi, T. Ishikawa, Y. Kurihara, Y. Shimizu, F. Yuasa. 2010. 9 pp. PoS ACAT2010 (2010) 073

7.Numerical Evaluation of Feynman Integrals by a Direct Computation Method F. Yuasa, T. Ishikawa, J. Fujimoto, N. Hamaguchi, E. de Doncker, Y. Shimizu. Apr 2009. 5 pp. PoS ACAT08 (2008) 122
8. Status reports from the GRACE group Y. Yasui, T. Ueda, E. de Doncker, J. Fujimoto, N. Hamaguchi, T. Ishikawa, Y. Shimizu. F. Yuasa Oct 2007. 6 pp. eConf C0705302 (2007) LOOP04 (LCWS-2007-LOOP04)
9. Precise Numerical Evaluation of the Scalar One-Loop Integrals with the Infrared Divergence、 F. Yuasa E. de Doncker, J. Fujimoro, N. Hamaguchi, T. Ishikawa, Y. Shimizu Sep 2007. 10 pp. PoS ACAT2007 (2007) 087

国内研究会論文等

10. 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、石川 正、湯浅 富久子
情報処理学会 研究報告ハイパフォーマンスコンピューティング（HPC）,2015-HPC-152,(2015-12-16)
11. 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
情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC）,2014-HPC-147(8),1-8 (2014-12-02)
12. 量子モンテカルロ法による物性スペクトル計算への多倍長演算の適用
濱口信行 , 石川正 , 岩野薫
情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC）,2013-HPC-141(12),1-7 (2013-09-23）

13. SR16000/M1とBG/Qにおける性能，精度評価
濱口信行 , 石川正
情報処理学会研究報告ハイパフォーマンス
コンピューティング（HPC）,2012-HPC-136(6),1-5 (2012-09-26)
14. 数値解析から見たファインマンループ積分の特徴と
多倍長精度積分の適用
濱口　信行
京都大学数理解析研究所RIMS研究集会
「科学技術計算における理論と応用の新展開」
数理解析研究所講究録 1791:131-140, 2012
15. SR11000での多倍長演算使用例
濱口　信行
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. 2, (3), pp. 102-108,2009.
16. SR11000モデルK1における多倍長ルーチンによる数値積分
の高速化
濱口信行
情報処理学会情報処理学会研究報告ハイパフォーマンス
コンピューティング（HPC）,2008(125(2008-HPC-118)),
109-114 (2008-12-09)
17. 二重指数関数型積分法の素粒子物理学への応用
湯浅 富久子 , 濱口 信行
情報処理学会情報処理学会研究報告ハイパフォーマンス
コンピューティング（HPC）,2008(19(2008-HPC-114)),
31-36 (2008-03-05)

18. Baileyアルゴリズムによる多倍長演算の性能評価
    石川 正 , 濱口 信行
    情報処理学会情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC）,2008(19(2008-HPC-114)), 25-30 (2008-03-05)
19. 理論物理学計算への多倍長ライブラリの適用
    濱口信行（日立）
    情報処理学会情報処理学会研究報告ハイパフォーマンスコンピューティング(HPC）,2007-HPC-113 (2007-12-07)