各種計算機基本性能調査

平成２４年度第4四半期

目次

1. はじめに

使用しました計算機の論理最大性能は
以下の様なものです.

SR16000/M1　1ノード　980.48GFLOPs
BG/Q　　1ノード　204.8GFLOPs

今年度のまとめも兼ね,simd機構,smt機構,並列化に関する基本性能測定結果をまとめました。

また並列化ソースを自動的に作成するツールも頻繁に使用する様になり,使用法とその効果、注意事項もまとめています。

BG/Qにおける姫野ベンチ,QCDの並列化に関しては,すべてParallel Program Generatorで作成した,スレッド並列用プログラムを使用しています。

2.Parallel Program Generator

ノード内でのsingle実行プログラム
複数ノード内でのフラットMPI実行プログラム

自動的にOpenmp指示行を挿入するツール。

ノード内でのスレッド並列実行プログラム

複数ノード内でのノード間はMPI、ノード内はスレッド並列のハイブリッド並列実行プログラム

シングル実行用プログラム　　(expq.f)

```
      sum1=0.0q0
      do i1=1,n
      xx=x30(i1)*cnt0
      by=1.0q0-xx
      cnt2=by-ay
      sum2=0.0q0
      do i2=1,n
      yy=x30(i2)*cnt2
      bz=1.0q0-xx-yy
      cnt4=bz-az
      sum3=0.0q0
      do i3=1,n
      zz=x30(i3)*cnt4
      d = -xx*yy*s-tt*zz*(1.0q0-xx-yy-zz)+(xx+yy)*ramda**2+
     1    (1.0q0-xx-yy-zz)*(1.0q0-xx-yy)*fme**2+zz*(1.0q0-xx-
           yy)*fmf**2
sum3=sum3+cnt0*cnt2*cnt4*gw30(i1)*gw30(i2)*gw30(i3)/d**2
      end do
      sum2=sum2+sum3*h
      end do
      sum1=sum1+sum2*h
      end do
      result=sum1*h
```

f90 expq.f -Os -loglist -c

とコマンドを投入しますと以下の様な情報が出力されます

```
**
**  Parallel processing starting at loop entry
**  Parallel function: _parallel_func_2_MAIN
**  Parallel loop
**    D: TLOCAL variable
**    ZZ: TLOCAL variable
**    SUM3: TLOCAL variable
**    CNT4: TLOCAL variable
**    YY: TLOCAL variable
**    SUM2: TLOCAL variable
**    CNT2: TLOCAL variable
**    XX: TLOCAL variable
**    SUM1: reduction variable (SUM)
**    I3: TLOCAL variable
**    I2: TLOCAL variable
      do i1=1,n
      xx=x30(i1)*cnt0
      by=1.0q0-xx
      cnt2=by-ay
      sum2=0.0q0
**
      do i2=1,n
      yy=x30(i2)*cnt2
      bz=1.0q0-xx-yy
      cnt4=bz-az
      sum3=0.0q0
```

次ページに続く

```
**  Continued parallel processing
**  Parallel processing finishing at loop exit
**
**    4 streams (GW30[]x2, X30[]x2) pre-fetch applied.
**
      do i3=1,n
      zz=x30(i3)*cnt4
      d = -xx*yy*s-tt*zz*(1.0q0-xx-yy-zz)+(xx+yy)*ramda**2+
     1    (1.0q0-xx-yy-zz)*(1.0q0-xx-yy)*fme**2+
     2     zz*(1.0q0-xx-yy)*fmf**2
      sum3=sum3+cnt0*cnt2*cnt4*gw30(i1)*gw30(i2)*gw30(i3)/d**2
      end do
      sum2=sum2+sum3*h
      end do
      sum1=sum1+sum2*h
      end do
```

これをもとに

./ppg　expq.f　とコマンドを投入するとスレッド並列実行用プログラムが,omp_expq.f に作成されます。

スレッド並列実行用プログラム　(omp_expq.f)

```
      sum1=0.0q0
C$OMP PARALLEL DO
C$OMP&  SCHEDULE(STATIC)
C$OMP&  PRIVATE(I1)
C$OMP&  PRIVATE(XX)
C$OMP&  PRIVATE(BY)
C$OMP&  PRIVATE(CNT2)
C$OMP&  PRIVATE(SUM2)
C$OMP&  PRIVATE(I2)
C$OMP&  PRIVATE(YY)
C$OMP&  PRIVATE(BZ)
C$OMP&  PRIVATE(CNT4)
C$OMP&  PRIVATE(SUM3)
C$OMP&  PRIVATE(I3)
C$OMP&  PRIVATE(ZZ)
C$OMP&  PRIVATE(D)
C$OMP&  REDUCTION(+:SUM1)
      do i1=1,n
      xx=x30(i1)*cnt0
      by=1.0q0-xx
      cnt2=by-ay
      sum2=0.0q0
      do i2=1,n
      yy=x30(i2)*cnt2
      bz=1.0q0-xx-yy
      cnt4=bz-az
      sum3=0.0q0
```

次ページに続く

前ページからの続き

```
      do i3=1,n
      zz=x30(i3)*cnt4
      d = -xx*yy*s-tt*zz*(1.0q0-xx-yy-zz)+(xx+yy)*ramda**2+
     1    (1.0q0-xx-yy-zz)*(1.0q0-xx-yy)
     2   *fme**2+zz*(1.0q0-xx-yy)*fmf**2
      sum3=sum3+cnt0*cnt2*cnt4*gw30(i1)*gw30(i2)*
     1 gw30(i3)/d**2
      end do
      sum2=sum2+sum3*h
      end do
      sum1=sum1+sum2*h
      end do
C$OMP END PARALLEL DO
      result=sum1*h
```

性能確認用ジョブ　SR16000/M1　1ノードで実行

inf *ra* *box*

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

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

$$s = -500^2, t = -150^2, m_e = 0.0005, m_f = 150, \lambda = 10^{-30}$$

二重指数関数型積分, 分点数1024

シングル実行
size=　　1024
result= 0.3561736882133282 6E-006
elapse=　129.020167350769043　　sec

64スレッド実行(自動並列)
size=　　1024
result= 0.3561736882133282 6E-006
elapse=　2.3954825401306152 3　　sec

64スレッド実行(作成したプログラム)
size=　　1024
result= 0.3561736882133282 6E-006
elapse=　2.3916270732879638 7　　sec

並列化効果が出ています。
OPENMP指示行の挿入の参考として使用できる.
他のスレッド並列をサポートしている計算機で
**　　実行可能。**

フラットmpi実行プログラムから（expq.mpi.f）ハイブリッド並列実行プログラム（omp_expq.mpi.f）の作成

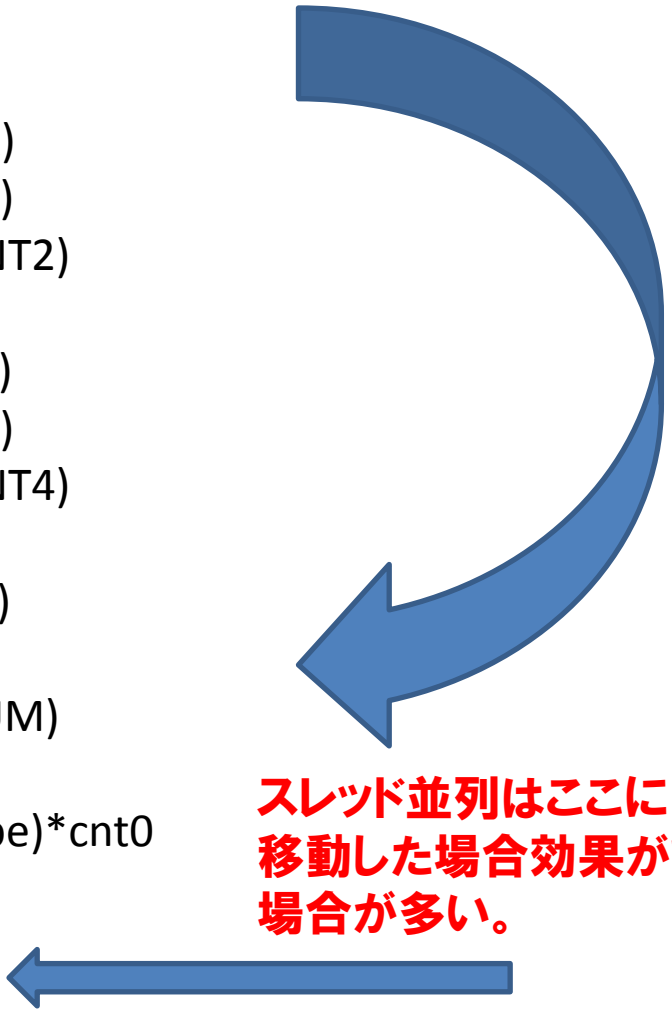
フラットMPI実行プログラム

```
call mpi_barrier(mpi_comm_world,ierr)
sec1 = mpi_wtime()
do i1=1,n/npe
xx=x30(i1+id*n/npe)*cnt0
by=1.0q0-xx
cnt2=by-ay
do i2=1,n
yy=x30(i2)*cnt2
bz=1.0q0-xx-yy
cnt4=bz-az
do i3=1,n
zz=x30(i3)*cnt4
d = -xx*yy*s-ttt*zz*(1.0q0-xx-yy-zz)+(xx+yy)*ramda**2+
1    (1.0q0-xx-yy-zz)*(1.0q0-xx-yy)*fme**2+zz*(1.0q0-xx-yy)
2   *fmf**2
w3(i3)=cnt0*cnt2*cnt4*gw30(i1+n*id/npe)
1*gw30(i2)*gw30(i3)/d**2
end do
   :
   :
```

./ppg expq.mpi.f　コマンド投入

作成されたスレッド並列実行プログラム

```
      call mpi_barrier(mpi_comm_world,ierr)
      sec1 = mpi_wtime()
C$OMP PARALLEL
C$OMP DO
C$OMP&  SCHEDULE(STATIC)
C$OMP&  PRIVATE(W2)
C$OMP&  PRIVATE(W3)
C$OMP&  PRIVATE(I)
C$OMP&  PRIVATE(I1)
C$OMP&  PRIVATE(XX)
C$OMP&  PRIVATE(BY)
C$OMP&  PRIVATE(CNT2)
C$OMP&  PRIVATE(I2)
C$OMP&  PRIVATE(YY)
C$OMP&  PRIVATE(BZ)
C$OMP&  PRIVATE(CNT4)
C$OMP&  PRIVATE(I3)
C$OMP&  PRIVATE(ZZ)
C$OMP&  PRIVATE(D)
C$OMP&  PRIVATE(SUM)
      do i1=1,n/npe
      xx=x30(i1+id*n/npe)*cnt0
      by=1.0q0-xx
      cnt2=by-ay
      do i2=1,n
      yy=x30(i2)*cnt2
      bz=1.0q0-xx-yy
            :
            :
```



スレッド並列はここに移動した場合効果がでる場合が多い。

検証用に使用した問題

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

$\lambda = 10^{-60}$, 解析近似解 $= 0.68294881938740 27\mathrm{d}-06$

使用計算機

BG／Q

- **none　フラットMPI**
- **outer　最外側でスレッド並列**
- **inner　外側2番目でスレッド並列**

二重指数関数型積分,分点数4096
倍精度演算16016GFLOPに相当
（性能モニターより）

実行時間一覧表(秒)　(BG/Q)

ノード数	mpi数	smp数	ハイブリッド	実行時間
32	512	1	none	60.892
32	1024	1	none	32.606
32	2048	1	none	22.795
32	32	16	outer	62.828
32	32	32	outer	35.15
32	32	64	outer	22.585
32	32	16	inner	62.714
32	32	32	inner	34.212
32	32	64	inner	22.462
128	128	16	inner	15.679
128	128	32	inner	8.572
128	128	64	inner	5.628
128	128	16	outer	15.707
128	128	32	outer	8.788
128	128	64	outer	8.788
512	512	16	inner	3.92
512	512	32	inner	2.157
512	512	64	inner	1.415
512	512	16	outer	7.854
512	512	32	outer	7.854
512	512	64	outer	7.854

*mpi*数×*smp*数≤分点数(今回4096)なら方式上の差異はない.

一般にinnerのハイブリッド並列が最も性能が良くなります。

3.SR16000/M1 システム

3.1 simd テスト

(1) 複素数型乗算

```
*poption parallel,tlocal(a1,a2,b1,b2,c1,c2)
      do j=1,n
      do i=1,n,nn
      do ii=i,i+nn-1
      a1(ii-i+1)=a(1,i,j)
      a2(ii-i+1)=a(2,i,j)
      b1(ii-i+1)=b(1,i,j)
      b2(ii-i+1)=b(2,i,j)
      end do
*soption unroll(2)
      do ii=1,nn
      c1(ii)=a1(ii)*b1(ii)-a2(ii)*b2(ii)
      c2(ii)=a1(ii)*b2(ii)+a2(ii)*b1(ii)
      end do
      do ii=1,nn
      c(1,i+ii-1,j)=c1(ii)
      c(2,i+ii-1,j)=c2(ii)
      end do
      end do
      end do
```

simd化

(2)実数型4倍精度加算

```
*poption parallel,tlocal(a1,a2,b1,b2,c1,c2,t1,t2,t3,t4,t5,t6)
      do j=1,n
      do i=1,n,nn
      do ii=i,i+nn-1
      a1(ii-i+1)=a(1,i,j)
      a2(ii-i+1)=a(2,i,j)
      b1(ii-i+1)=b(1,i,j)
      b2(ii-i+1)=b(2,i,j)
      end do
*soption unroll(2)
      do ii=1,nn
      t1=a1(ii)+b1(ii)
      t2=t1-a1(ii)
      t3=(a1(ii)-(t1-t2))+(b1(ii)-t2)
      t4=t3+a2(ii)+b2(ii)
      t5=t1+t4
      t6=t5-t1
      c1(ii)=t5
      c2(ii)=(t1-(t5-t6))+(t4-t6)
      end do
      do ii=1,nn
      c(1,i+ii-1,j)=c1(ii)
      c(2,i+ii-1,j)=c2(ii)
      end do
      end do
      end do
```

simd化

(3)実数型4倍精度乗算

```
*poption parallel,tlocal(a1,a2,b1,b2,c1,c2,t1,t2,t3,t4,t5,
       p1,a11,a12,p2,b11,b12)
     do j=1,n
     do i=1,n,nn
     do ii=i,i+nn-1
     a1(ii-i+1)=a(1,i,j)
     a2(ii-i+1)=a(2,i,j)
     b1(ii-i+1)=b(1,i,j)
     b2(ii-i+1)=b(2,i,j)
     end do
*soption unroll(2)
       do ii=1,nn
       t1=a1(ii)*b1(ii)
      p1=r*a1(ii)
      a11=a1(ii)-p1
      a11=a11+p1
      a12=a1(ii)-a11
      p2=r*b1(ii)
      b11=b1(ii)-p2
      b11=b11+p2
      b12=b1(ii)-b11
      t2=(((a11*b11-t1)+a11*b12)+a12*b11)+a12*b12
     t3=t2+a1(ii)*b2(ii)
     t4=t3+a2(ii)*b1(ii)
     t5=t1+t4
     c1(ii)=t5
     c2(ii)=t4-(t5-t1)
     end do
```

次ページに続く

前ページからの続き

```
do ii=1,nn
c(1,i+ii-1,j)=c1(ii)
c(2,i+ii-1,j)=c2(ii)
end do
end do
end do
```

simd化

simd 化　テスト結果一覧表

nn=256,*soption unroll(2)を使用
—Os　実行時間はすべて秒

（1）複素数変数乗算　C*16　n=65536

smp数	オリジナル	SIMD化
1	20.553	13.916
2	10.355	6.966
4	5.346	3.507
8	3.469	1.87
16	1.771	1.139
32	0.831	0.457
64	0.93	0.501

（2）実数変数4倍精度加算　R*16　n=32768

smp数	オリジナル	SIMD化
1	6.686	3.955
2	3.349	1.981
4	1.681	0.997
8	0.859	0.516
16	0.43	0.257
32	0.215	0.129
64	0.271	0.136

（3）実数変数4倍精度乗算　R*16　n=32768

smp数	オリジナル	SIMD化
1	5.529	5.219
2	2.793	2.618
4	1.44	1.361
8	0.833	0.675
16	0.417	0.336
32	0.209	0.168
64	0.231	0.154

Simd化の効果が見られます。

3.2　複素数型内積演算

ループ長 $= 2^{20}$, 演算量 $= 8796GFLOP$(ノード数に依存しない)

smp　オリジナルソース

```
*poption noparallel
*soption unroll(1)
      do iter=1,loop
      sum=dcmplx(0.0d0,0.0d0)
*poption parallel
      do i=1,n
      sum=sum+a(i)*b(i)
      end do
      end do
```

smp　simd化ソース

```
*poption noparallel
*soption unroll(1)
      do j=1,loop
      tr=0.0d0
      ti=0.0d0
*poption parallel
      do i=1,n
      tr=tr+ar(i)*br(i)+ai(i)*bic(i)
      ti=ti+ar(i)*bi(i)+ai(i)*br(i)
      end do
      sum=dcmplx(tr,ti)
      end do
```

ハイブリッド並列　オリジナルソース

```
*poption noparallel
*soption unroll(1)
    do j=1,loop
*poption parallel
    do i=1,n/npe
    ss(id)=ss(id)+a(i+id*n/npe)*b(i+id*n/npe)
    end do
    end do
```

ハイブリッド　simd化ソース

```
*poption noparallel
*soption unroll(1)
    do j=1,loop
*poption parallel
    do i=1,n/npe
    s1=s1+ar(i+id*n/npe)*br(i+id*n/npe)+ai(i+id*n/npe)*bic(i+id*n/npe)
    s2=s2+ar(i+id*n/npe)*bi(i+id*n/npe)+1 ai(i+id*n/npe)*br(i+id*n/npe)
    end do
    end do
```

複素数内積演算性能測定結果

実行時間一覧表(秒)			
ノード数	smp数(ノード当たり)	オリジナル	simd化
1	32	57.665	65.031
1	64	110.909	60.591
4	32	26.934	27.099
4	64	13.721	14.592
8	32	13.461	13.563
8	64	6.827	7.273

1ノード
　　オリジナルでは,smt=off,simd化では
　　smt=onが有効。
複数ノード
　　smt=on が有効。smt=off の2倍近い性能。

3.3　ルジャンドル変換とFFT計算

球座標系でのスペクトル法での計算で使用されるルジャンドル変換計算とFFT計算用のテストプログラムを作成して行った性能測定結果です。

ルジャンドル変換計算は形は三角行列の行列積となるため行列積計算での用語より,基本型,内積型,外積型としています。演算量は性能モニターの値を使用しています。

FFT計算は基底を,2,3,4,5,一般とした基本型と2,3,4,5,6,7,8,9,11,12,16とした応用型に分けています。演算量は基本型での性能モニターの値を使用しています。応用型ではチューニングも含まれていますので性能の絶対値は大きくでるケースもあります。

ルジャンドル変換　GFLOPs一覧表

タイプ	1 node	4 node	8 node
基本型	128.214	368.685	1022.808
内積型	419.536	1649.487	3332.407
外積型	239.288	857.066	1910.665

内積型の性能が良い。

FFT　GFLOPs　一覧表

N	1node		4node		8node	
	基本型	応用型	基本型	応用型	基本型	応用型
3^6	426	803	1762	3028	2912	4961
4^6	141	142	477	476	1090	1079
5^5	180	286	677	1058	1385	2182
6^4	364	653	1189	2143	2509	4077
7^4	60	268	193	999	463	2023
8^4	143	142	472	477	1114	1106
9^3	387	784	1866	3034	1775	5752
10^3	512	955	2102	3238	3620	5774
12^3	232	494	882	1757	1756	3527
16^3	142	143	470	457	1102	1105
11^3	134	134	513	506	1043	979

Nが3,5,6,7,9,10,12のべき乗の場合、応用型の効果が大きい。

3.4　姫野ベンチ

演算量は単精度、倍精度はプログラム内の値を使用し,複素数型,4倍精度は性能モニターの値を採用しました。

姫野ベンチ測定結果

サイズ2049×2049×1025

分割	8×8×8	*flat*	
	2×2×2	*hybrid*	8*mpi*×64*smp*
	4×4×4	*hybrid*	64*mpi*×8*smp*

GFLOPs一覧表　8ノード

ソース	mpi	単精度	倍精度	複素数	4倍精度
ori	2*2*2	227	350	629	940
ori	4*4*4	259	480	862	968
ori	flat	269	500	895	977
tune	2*2*2	560	315	576	942
tune	4*4*4	769	444	797	932
tune	flat	805	473	834	983

性能的には,flat　mpiが良い値となっています。

3.5 QCD

1 ノード,自動並列実行

SR16000/M1 実行結果一覧表　(単位 秒)

data	16*16*16*32	24*24*24*48	32*32*32*64
time for Uinit	0.042	0.213	2.29
D_W mult	2736	2576	3650
total	223.87	861.28	3779.2
solver	223.65	860.24	3774.3
cg-step	220.55	847.24	3733.8
mult	176.6	658.3	2887.5

Single 実行

data 16*16*16*32

smp数	1smp	16smp	32 smp	64 smp
total	221.47	18.759	11.896	14.914
solver	221.26	18.354	11.686	14.708
cg-step	218.17	18.013	11.421	14.402
mult	175.19	13.672	8.952	11.599

data 24*24*24*48

smp数	1smp	16smp	32 smp	64 smp
total	861.36	155.09	87.848	77.041
solver	859.67	153.87	86.804	75.973
cg-step	846.68	151.38	85.262	74.631
mult	657.86	119.36	72.218	60.304

data 32*32*32*64

smp数	1smp	16smp	32 smp	64 smp
total	3760.4	508.39	339.15	396.52
solver	3754.9	503.22	336.02	393.14
cg-step	3714.7	497.45	332.08	388.36
mult	2867.7	388.25	266.41	299.02

自動並列化オーバーヘッドが小さい。
32smp (smt off)の場合が最も性能が良い。

4. BG/Q システム

4.1 simd テスト

1 Node

BG/Q 性能測定結果

倍精度複素数演算 $d = a + b \times c$

サイズ 8192 *simd* 用バッファサイズ 512 400回実行

4倍精度実数演算 $c = a \times b$

サイズ 16384 *simd* 用バッファサイズ 1024 100回実行

実行時間一覧表(秒)

タイプ	ソース	16smp	32smp	64smp
4倍精度	オリジナル	45.358	45.283	45.107
	simd化	26.864	26.717	26.791
複素数	オリジナル	62.892	63.065	63.209
	simd化	26.712	26.745	26.617

16smp,32smp,64smp で性能差なし。
Simd化の効果は大きい。

4.2 複素数内積演算

BG/Q 性能測定結果

ループ長 $= 2^{20}$, 演算量 $= 8796GFLOP$(ノード数に依存しない)

ハイブリッド並列 実行時間一覧表

ノード数	smp数	実行時間(秒)
32	16	39.984
32	32	21.166
32	64	16.206
128	16	9.502
128	32	5.3
128	64	4.087
512	16	2.379
512	32	1.325
512	64	1.032

smtの効果

smt=2/smt=1 1.8

smt=4/smt=1 2.3

Smtの効果はsimd/nosimdの効果と同じ様な値となっています。

4.3 ルジャンドル変換とFFT計算

BG/Q 32ノード		ルジャンドル変換計算	
GFLOPs 一覧表			
タイプ	64smp	32smp	16smp
基本型	615	713	520
内積型	566	358	192
外積型	1104	790	709

外積型の性能が最も良く,smtの効果も大きい。

FFT GFLOPs 一覧表

N	基本型			応用型		
	64smp	32smp	16smp	64smp	32smp	16smp
3^6	629	1264	1164	1277	1557	1115
4^6	588	565	587	582	565	583
5^5	696	732	759	1101	1069	1077
6^4	595	622	1077	1097	959	787
7^4	154	168	205	975	925	918
8^4	598	575	600	590	572	594
9^3	635	1034	1146	1285	1476	1114
10^3	789	1008	1455	1548	1476	1241
12^3	704	715	910	1276	1080	732
16^3	598	576	603	590	574	596
11^3	181	228	316	182	232	317

smtの効果は,smt=2が良い値となっています。
応用型では基底3,5,7,9の奇数基底の効果が大きい。

4.4　姫野ベンチ

測定条件一覧表

small	サイズ1025×513×513(32ノード)
hybrid	4×4×2
flat	16×16×8
middle	サイズ1025×1025×1025(128ノード)
hybrid	8×4×4
flat	32×16×16
large	サイズ2049×2049×1025(512ノード)
hybrid	8×8×8
flat	32×32×32

演算量　単精度、倍精度　プログラム内記述の値
複素数,4倍精度　SR16000/M1 の
性能モニターの値

姫野ベンチGFLOPs一覧表

（単精度）

サイズ	ソース	mpi	GFLOPs	サイズ	ソース	mpi	GFLOPs
small	ori	hybrid	236	small	ori	hybrid	135
	ori	flat	271		ori	flat	146
	tune	hybrid	247		tune	hybrid	142
	tune	flat	314		tune	flat	177
middle	ori	hybrid	770	middle	ori	hybrid	451
	ori	flat	957		ori	flat	497
	tune	hybrid	795		tune	hybrid	479
	tune	flat	1007		tune	flat	582
large	ori	hybrid	3767	large	ori	hybrid	2325
	ori	flat	4274		ori	flat	1691
	tune	hybrid	4055		tune	hybrid	2361
	tune	flat	4746		tune	flat	2615

（倍精度）

（複素数）

サイズ	ソース	mpi	GFLOPs	サイズ	ソース	mpi	GFLOPs
small	ori	hybrid	227	small	ori	hybrid	589
	ori	flat	193		ori	flat	577
	tune	hybrid	252		tune	hybrid	633
	tune	flat	293		tune	flat	698
middle	ori	hybrid	781	middle	ori	hybrid	2182
	ori	flat	711		ori	flat	2232
	tune	hybrid	848		tune	hybrid	2342
	tune	flat	1021		tune	flat	2674
large	ori	hybrid	3686	large	ori	hybrid	9465
	ori	flat	3049		ori	flat	9205
	tune	hybrid	4204		tune	hybrid	10186
	tune	flat	4593		tune	flat	11077

（4倍精度）

並列化手法では、flat MPI とMPIとsmpのハイブリッド並列では大差はない。

4.5　QCD

1ノード実行
smpソースはparallel program generator で作成

BG/Q QCD 実行結果一覧表　（単位 秒）

data	16*16*16*32	24*24*24*48	32*32*32*64	
time for Uinit	0.56	2.81	8.24	
D_W mult	2744	2574	3654	
total	1295.1	6212.8	27510	Single 実行
solver	1293.9	6206.9	27492	
cg-step	1275.9	6144.7	27204	
mult	1078.6	5183.2	22963	

data 16*16*16*32

smp数	1smp	16 smp	32 smp	64smp
total	1381.1	101.33	87.2	123.13
solver	1379.9	100.38	86.25	122.14
cg-step	1359.8	98.88	84.91	120.24
mult	1177.4	80.53	66.73	96.65

data 24*24*24*48

smp数	1smp	16 smp	32 smp	64smp
total	6532.6	508.5	491.87	512.21
solver	6526.8	503.84	487.21	507.53
cg-step	6427.3	496.11	479.78	499.84
mult	5605.6	382.66	366.59	381.77

data 32*32*32*64

smp数	1smp	16 smp	32 smp	64smp
total	28954	2347.4	2221.9	2247.6
solver	28936	2332.3	2207	2232.5
cg-step	28624	2307.6	2183.6	2208.9
mult	24922	1794.7	1673.4	1694.3

smp並列化オーバーヘッドが約5%と大きい。
16smp　(smt=1)で性能が飽和している。