各種計算機基本性能調査

平成24年度第3四半期

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
xm1　　1ノード　844.8GFLOPs

　xm1では今回論理コア数が128に拡大されたため,その効果を記述しています.

数値積分で使用しました問題の定義式をまとめて記述しました。

また並列化ソースを自動的に作成すろツールも
使用し始めました。

(1) 三次元積分

Infra box

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

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

(2) 四次元積分

s221

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

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

（3）五次元積分

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

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

laporta（D）

```
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
```

laporta（F）

```
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
```

2.分子動力学計算でのチューニング例

分子動力学計算及び重力多体問題でよく
行われている演算量削減のチューニングでは、
シングル実行では効果がありますが,
並列実行時には　オーバーヘッドによりあまり
性能がでない事が往々　にしておこります。

ここではloglistでの解析とそれを使用して
openmpソースの作成例を合わせて記載しました。

ソースに関しては,分子動力学計算のチューニング
したものは,重力多体問題のオリジナルソースと
なっています。

発端は,N=128でのSR16000/M1　8ノード
256MPI　3735秒,BG/Q　32ノード 1024MPI で
4170秒だったのが,SR16000/M1 1ノード
64smpで3680秒となったことから調査を
開始しました.

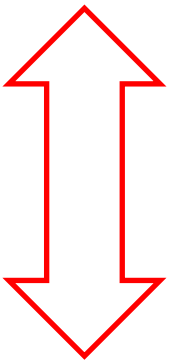
分子動力学計算のソース例

```
   fx=0.0d0
    fy=0.0d0
    do 100 i=1,n
    sumx=0.0d0
    sumy=0.0d0
    do 200 j=1,i-1
    r=sqrt((x(i)-x(j))**2*(y(i)-y(j))**2+eps*eps)
    sumx=sumx-dt*g*(x(i)-x(j))/r**3
    sumy=sumy-dt*g*(y(i)-y(j))/r**3
200 continue
    do 300 j=i+1,n
    r=sqrt((x(i)-x(j))**2*(y(i)-y(j))**2+eps*eps)
    sumx=sumx-dt*g*(x(i)-x(j))/r**3
    sumy=sumy-dt*g*(y(i)-y(j))/r**3
300 continue
    fx(i)=sumx
    fy(i)=sumy
100 continue
```

演算量を削減するためよく以下のチューニングが行われます。

```
      fx=0.0d0
      fy=0.0d0
      do 400 i=1,n-1
      sumx=0.0d0
      sumy=0.0d0
      do 500 j=i+1,n
      r=sqrt((x(i)-x(j))**2*(y(i)-y(j))**2+eps*eps)
      wx=-dt*g*(x(i)-x(j))/r**3
      wy=-dt*g*(y(i)-y(j))/r**3
      fx(j)=fx(j)-wx
      fy(j)=fy(j)-wy
      sumx=sumx+wx
      sumy=sumy+wy
  500 continue
      fx(i)=fx(i)+sumx
      fy(i)=fy(i)+sumy
  400 continue
```

SR16000/M1　1cpu　N=2^16=65536
　origin　32.971 sec
　tuning　17.859 sec



スレッド並列時には性能が逆転する.

SR16000/M1　1node　64smp　　N=2^18=262144
　origin　　13.534 sec
　tuning　　15.475 sec

MPI化した場合は更にその差は大きくなる。！

最適化ログリスト(origin)

```
**  Parallel processing starting at loop entry
**  Parallel function: _parallel_func_2_MAIN
**  Parallel loop (cyclic distribution)
**    R: TLOCAL variable
**    SUMY: TLOCAL variable
**    SUMX: TLOCAL variable
**    J: TLOCAL variable
**  Parallel processing finishing at loop exit
    do 100 i=1,n
    sumx=0.0d0
    sumy=0.0d0
```

```
**    [DO 200]
**    SWPL applied.
**    2 streams (X[], Y[]) pre-fetch applied.
**
**    Innermost loop unrolled (4 times).
**    SWPL applied.
**    1 streams (CompGen #5) pre-fetch applied.
**
**    Loop distributed for vector library: 5 pieces; 2 candidates for
library calls.
**    Simdization applied.
**    Innermost accumulator variables expanded (2 times).
**    SWPL applied.
**    3 streams (CompGen #5, X[], Y[]) pre-fetch applied.
**
**    Replace with call to PVP library (vsqrt).
**
**    Replace with call to PVP library (vrec).
```

Simdの適用,SWPL,高速ライブラリが使用されている.

```
**    [DO 300]
**    SWPL applied.
**    2 streams (X[], Y[]) pre-fetch applied.
**
**    Innermost loop unrolled (4 times).
**    SWPL applied.
**    1 streams (CompGen #7) pre-fetch applied.
**
**    Loop distributed for vector library: 5 pieces; 2
candidates for library calls.
**    Simdization applied.
**    Innermost accumulator variables expanded (2 times).
**    SWPL applied.
**    3 streams (CompGen #7, X[], Y[]) pre-fetch applied.
**
**    Replace with call to PVP library (vsqrt).
**
**    Replace with call to PVP library (vrec).
**
```

Simdの適用,SWPL,高速ライブラリが使用されている.

ログリスト参照からopenmpソース作成例

```
C$OMP PARALLEL DO
C$OMP&  SCHEDULE(STATIC,1)
C$OMP&  PRIVATE(I)
C$OMP&  PRIVATE(SUMX)
C$OMP&  PRIVATE(SUMY)
C$OMP&  PRIVATE(J)
C$OMP&  PRIVATE(R)
      do 100 i=1,n
      sumx=0.0d0
      sumy=0.0d0
      do 200 j=1,i-1
      r=sqrt((x(i)-x(j))**2*(y(i)-y(j))**2+eps*eps)
      sumx=sumx-dt*g*(x(i)-x(j))/r**3
      sumy=sumy-dt*g*(y(i)-y(j))/r**3
  200 continue
      do 300 j=i+1,n
      r=sqrt((x(i)-x(j))**2*(y(i)-y(j))**2+eps*eps)
      sumx=sumx-dt*g*(x(i)-x(j))/r**3
      sumy=sumy-dt*g*(y(i)-y(j))/r**3
  300 continue
      fx(i)=sumx
     fy(i)=sumy
  100 continue
C$OMP END PARALLEL DO
```

最適化ログリスト(tuning)

```
XX  Serial loop
**  --- Loop distributed for parallelization ---
**     FY: unknown loop dependency
**     FX: unknown loop dependency
**
**
**    [DO 400]
**    Simdization applied.
**    Innermost loop unrolled (2 times).
**    SWPL applied.
**    4 streams (CompGen #5, CompGen #6, FX[], FY[])
 pre-fetch applied.
**
```

DO 400 では並列化されていない.

```
**  Parallel loop
**    CompGen #14: reduction array (SUM)
**    CompGen #13: reduction array (SUM)
**    WY: TLOCAL variable
**    WX: TLOCAL variable
**    R: TLOCAL variable
**  --- Add barrier at loop entry ---
**  --- Add barrier at loop exit ---
**    [DO 500]
**    SWPL applied.
**    2 streams (X[], Y[]) pre-fetch applied.
**    Innermost loop unrolled (4 times).
**    SWPL applied.
**    1 streams (CompGen #10) pre-fetch applied.
**
**    Loop distributed for vector library: 5 pieces; 2 candidates
 for library calls.
**    Simdization applied.
**    Innermost accumulator variables expanded (2 times).
**    SWPL applied.
**    5 streams (CompGen #10, FX[], FY[], X[], Y[]) pre-fetch applied.
**
**    Replace with call to PVP library (vsqrt).
**
**    Replace with call to PVP library (vrec).
```

最内側DOループの並列化と最適化を実施.
オーバーヘッドは大となる.

```
C$OMP PARALLEL
 do 400 i=1,n-1
C$OMP MASTER
      sumx=0.0d0
      sumy=0.0d0
C$OMP END MASTER
C$OMP BARRIER
C$OMP DO
C$OMP&  SCHEDULE(STATIC)
C$OMP&  PRIVATE(J)
C$OMP&  PRIVATE(R)
C$OMP&  PRIVATE(WX)
C$OMP&  PRIVATE(WY)
C$OMP&  REDUCTION(+:SUMX)
C$OMP&  REDUCTION(+:SUMY)
      do 500 j=i+1,n
      r=sqrt((x(i)-x(j))**2*(y(i)-y(j))**2+eps*eps)
      wx=-dt*g*(x(i)-x(j))/r**3
      wy=-dt*g*(y(i)-y(j))/r**3
      fx(j)=fx(j)-wx
      fy(j)=fy(j)-wy
      sumx=sumx+wx
      sumy=sumy+wy
 C$OMP END DO NOWAIT
C$OMP BARRIER
C$OMP MASTER
      fx(i)=fx(i)+sumx
      fy(i)=fy(i)+sumy
C$OMP END MASTER
  400 continue
C$OMP END PARALLEL
```

ログリスト参照からopenmpソース作成例

3. SR16000/M1 システム

3.1 分子動力学計算

N=128 所要メモリ 224MB

従来版					
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	
smp数	ノード数	VdW	Coulomb	全体	
32	1	1769.65239	4245.32686	6014.98169	
64	1	1180.91332	2499.41767	3680.33458	

演算量は2倍
ハイブリッドの効果は小

修正版				
MPI数	ノード数	VdW	Coulomb	全体
128	4	4487.29192	5159.58867	9646.93067
256	4	1767.90486	3057.6814	4825.68794
256	8	2244.68742	2588.84542	4833.58705
512	8	885.04123	1528.38884	2413.52728
ハイブリッド版				
SMP数	ノード数	VdW	Coulomb	全体
32	4	3675.13414	4544.54538	8219.68586
64	4	2054.28677	3273.93475	5328.23547
32	8	1838.06506	2273.0442	4111.11608
64	8	1027.48181	1637.56081	2665.05695

3.2 4次元積分

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

$N=1152$　　演算量　47505　$GFLOP$

1 ノード	実行時間(秒)			
smp数	smt	実行時間(秒)	性能(GFLOPs)	実行効率(%)
32	off	147.387	322	32.8
64	on	178.875	266	27.1

Simdが適用されるため,smt=offの場合の方が性能が良い.また実行効率も30%を超えている.

3.3　五次元積分

laporta（f）

サイズ　N=128　　所要テーブルサイズ　　2KB
演算量　　3642GFLOP

1 ノード	実行時間(秒)			
smp数	smt	実行時間(秒)	性能(GFLOPs)	実行効率(%)
32	off	11.674	312	31.8
64	on	7.909	460	46.9

テーブルサイズが小さいため,実行効率は良い.
Simdが適用されないため,smt　on の場合のほうが性能が良い.

4　BG/Q システム

4.1　分子動力学計算

BG/Qシステムにおいては,所要メモリ量に性能が大きく左右されます.!

(1) フラットMPI実行の場合

MPIオーバーヘッドを減らす場合,演算量だけでなく所要メモリ量も倍になります.。(他システムとのmpi_allgatherの仕様の差のため.)

**このため、所要メモリサイズが大きい場合,
修正ソースでは実行可能なMPI数が減ります.
実行可能なサイズでは,もとは,smt=2が最速ですが,修正ソースではsmt=4が最速となります.
(MPIオーバーヘッドが削減されると,simdが適用されない効果がみえてくるため.)**

(2) ハイブリット実行の場合

MPIオーバーヘッド削減のソースでハイブリッド実行をすると,メモリの制限もなくなり,効果が大きくなります.所要メモリ量が小さい場合,10倍以上の性能向上が得られています。

		N=48	(11.8125MB)	
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

修正版

MPI数	ノード数	VdW	Coulomb	全体
512	32	5.49513	8.59571	14.11402
1024	32	2.99941	4.56508	7.61897
2048	32	1.96861	2.87183	5.11686

ハイブリッド版

smp数	ノード数	VdW	Coulomb	全体
16	32	3.63762	5.71114	9.36463
32	32	1.87721	2.94602	4.83891
64	32	0.97889	1.71597	2.71098

所要メモリ小だとハイブリッドの効果大

		N=64	(28.0MB)	
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

修正版

MPI数	ノード数	VdW	Coulomb	全体
512	32	30.5287	48.11573	78.70039
1024	32	16.22005	25.30125	41.86008
2048	32	9.90448	15.39024	26.00268

ハイブリッド版

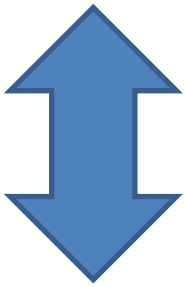
smp数	ノード数	VdW	Coulomb	全体
16	32	20.46678	31.77036	52.26934
32	32	10.52987	16.73987	27.30197
64	32	5.43206	9.65071	15.11501

		N=80	(54.6875MB)	
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
修正版				
MPI数	ノード数	VdW	Coulomb	全体
512	32	115.9476	183.24885	299.47552
1024	32	61.06849	95.75686	157.45681
2048	32	36.28185	57.77221	95.30769
ハイブリッド版				
smp数	ノード数	VdW	Coulomb	全体
16	32	77.67598	122.27164	200.00683
32	32	39.95052	63.00466	103.0142
64	32	20.81901	36.65174	57.53005

MPIオーバーヘッド削減が所要メモリ容量の微妙に影響をあたえはじめています。

		N=96	(94.5MB)	
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
修正版				
MPI数	ノード数	VdW	Coulomb	全体
512	32	235.6506	359.7902	595.97801
1024	32	120.71485	186.07948	307.88298
ハイブリッド版				
smp数	ノード数	VdW	Coulomb	全体
16	32	232.7985	361.71276	594.61322
32	32	119.54831	186.49663	306.14688
64	32	61.61527	109.81348	171.53069

		N=112	(150.0625MB)	
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
修正版				
MPI数	ノード数	VdW	Coulomb	全体
512	32	856.15347	1376.07514	2233.1181
ハイブリッド版				
smp数	ノード数	VdW	Coulomb	全体
16	32	584.54206	920.53324	1505.23899
32	32	299.31385	473.062223	772.53978
64	32	156.61686	271.30716	428.08792



ハイブリッド方式以外では実行しない方が良い事がわかります。

		N=128	(224MB)	
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
修正版				
MPI数	ノード数	VdW	Coulomb	全体
512	32	1906.68062	3070.87981	4978.95545
ハイブリッド版				
smp数	ノード数	VdW	Coulomb	全体
16	32	1307.681	2032.29291	3340.22335
32	32	668.91654	1050.9015	1720.0675
64	32	346.72815	604.46585	951.44338

4.2　五次元積分

laporta（f）

サイズ　N=128　　所要テーブルサイズ　　2KB
演算量　　3642GFLOP

コンパイルオプション

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	

コンパイルオプションで３－４倍の性能差がある.
最速はフラットMPI　2048MPI で
2825GFLOPs,実行効率43．1％

5　xm1 システム

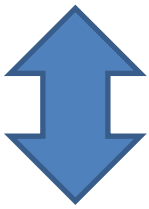
従来の最大論理コア数６４から１２８になりましたので今回、smtの効果を中心にまとめました。

5.1　分子動力学計算

論理コアが１２８になりましたのでBG／Qとの
比較を考慮して,96SMPが容易に実行できるように
サイズはN=96,所要メモリ９４．５MBで
実行しています。

従来のSMP用ソースをmdcore,
MPIオーバーヘッド削減の
Smp版をmdcore2と記しています.

mdcore				
smp数	ノード数	VdW	Coulomb	全体
32	1	370.355	1031.531	1402.025
64	1	246.078	611.013	857.267
96	1	223.676	446.46	670.328
128	1	173.189	340.836	514.222
mdcore2				
smp数	ノード数	VdW	Coulomb	全体
32	1	353.256	930.879	1254.276
64	1	306.571	605.227	911.972
96	1	229.255	442.445	671.89
128	1	177.715	339.45	517.357



Smpだと演算量が倍増しているのに実行時間は大差はありません.また128論理コアはともに効果大.

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

5.2 四次元積分

$S^{221}(-1;100,100,0,0,100)$　　$N = 576$

二次元積分　　*GPU*等で性能を向上させるために4重*DO*ループを2重*DO*ループ化したソース

四次元積分　　従来の4重*DO*ループのソース

64MPI で実行した場合の実行時間
2次元積分 25秒　4次元積分 15秒

＝＞GPU等とはチューニング方法が異なる事を示しています。

N=1152 演算量 47505GFLOPでの実行
実行ソースは4次元積分用。

32SMP	247秒
64SMP	278秒
96SMP	274秒
128SMP	206秒

SIMDが適用され、テーブルサイズ18KB.
このため複数の要因がからんだ結果となっています。
最速の実行効率は27．3％とそこそこの値です。

5.3　五次元積分

サイズ N=128,テーブルサイズ　256KB

演算量　laporta　(d)　2668GFLOP
laporta　(f)　3642GFLOP

実行時間一覧表(秒)			
プログラム	32smp	64smp	128smp
laporta(d)	30.005	31.151	19.369
laporta(f)	40.911	38.31	24.677

Smt=4の効果はともにありますが,実行効率は17％前後とさほど高くありません.5重DOループを2重DOループ化しているためテーブルサイズが大きくなり、メモリ性能が悪いxm1でその影響がでています。

SR16000／M1　と比較すると論理最大性能は16％低いだけですが実行性能は2．5－3倍程度低下しています。

5.4　QCD計算

サイズ　　128*8*8*16

所要メモリ　　79．4MB

実行性能(GFLOPs)一覧表

ソースタイプ	32smp	64smp	128smp
実数型	126	114	153
複素数型	102	96	128

64smp は以前はsmt off で今回smt on になったことによりオーバーヘッドが増えた事による現象をしめしています．
実行効率は128SMPで１８％となり,このサイズではよく出ているといえます。