各種計算機基本性能調査

平成25年度第2四半期

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

扱ったプログラムは
量子モンテカルロ法による物性スペクトル計算
に現れるGREEN関数計算で,今四半期の間に
10倍精度演算において大きな性能改善が
達成できました。

また,Parallel　Program Generator は
上記10倍精度演算での性能向上に寄与し、
サーバー系での並列化ではSR16000/M1 自動
並列以外はBG/Qも含め全てPpgenで作成した
ソースを使用しています。

量子モンテカルロ法では,特に記述がなければ
N=100,L=448
6倍精度　β=10,u=6
8倍精度　β=20,u=8
10倍精度　β=20,u=10
の条件で実行している。

行列積,QDR積はN=100,実行回数1000回
の実行時間(秒)を測定している。

2. SR16000/M1　システム

2.1　性能低下要因の解析と対策

前四半期実行時間一覧表(秒)

精度	32core	64smp
6倍精度	28.837	23.743
8倍精度	51.739	42.074
10倍精度(QDR分割なし)	733	783
10倍精度(QDR分割あり)	394	410

10倍精度演算の性能は,6倍精度演算,8倍精度演算の性能に比べ悪い。

原因:演算量が多くて最適化が行われず、
並列化がされなかった。

QDR積の計算

```
 g(:,:,mt,ispin)=0.q0
c
      do k=1,n
      do j=1,n
      do i=1,n
        g(i,j,mt,ispin)=g(i,j,mt,ispin) + q(i,k)*dd2(k)*dm(k,j)
      enddo
      enddo
      enddo
c
```

この計算を8倍精度,10倍精度サブルーチンを作成しコンパイルしますと,8倍精度ルーチンは並列化され,10倍精度ルーチンでは並列化されません。

使用したコンパイルオプションは
-W0,'opt(o(s),disbraket(0),uinline(2))'

opt(allocinline(0),approx(1),approxlib(0),array_overlap(1),
arraycomm(1),autocom(0),autoinline(0),autoinline_diag(0),
autoinline_target(0),contarea(0),convcheck(0),dblprec(0),
disbracket(0),divmove(1),divopt(0),expmove(1),fma(1),
force_soption(0),ifconv(normal),ifswpl(0),invariant_if(1),
ischedule(3),listvec(0),loopdiag(0),loopdistribute(1),
loopexpand(1),loopfuse(1),loopinnerdistribute(0),
loopinterchange(1),loopouterunroll(1),loopreroll(1),looptiling(0),
mathinline(0),o(4),prefetch(1),prefetch_zero(0),prod(1),profile(0),
pvfunc(0),quadlib(0),rapidcall(1),restrict(0),roughpow(0),
roughquad(0),roundup(0),scope(1),simd(normal),simd_expand(0),
swpl(1),uinline(2),workarray(1))

でscope(1)となっています。

このときのloflistは次ページ以降のようになっています。

8倍精度ルーチン

```
 subroutine qdrmul8(aa,dd,bb,cc,n)
      implicit real*8 (a-h,o-z)
      data r/134217729.0d0/
      real*8  aa(4,n,n),dd(4,n),bb(4,n,n),cc(4,n,n)
      real*8  a(4),b(4),c(4),s(4)
**
**  Parallel processing starting at loop entry
**  Parallel function: _parallel_func_28_QDRMUL8
**  Parallel loop
      do j=1,n
**
**
**    IF test is invarient in loop so moved to outside.
**    Innermost loop resolved (2 times).
**    SWPL applied.
**
      do i=1,n
**
      s=0.0d0
**
**
**    Innermost loop resolved (2 times).
**    2 streams (BB[], DD[]) pre-fetch applied.
**
      do k=1,n
```

並列化されています。

10倍精度ルーチン

```
 subroutine qdrmul10(aa,dd,bb,cc,n)
     implicit real*8 (a-h,o-z)
     data r/134217729.0d0/
     real*8  aa(5,n,n),dd(5,n),bb(5,n,n),cc(5,n,n)
     real*8  a(5),b(5),c(5),s(5)
**
**    many floating point operations: scope separation occured
XX    loop not optimized.
**
     do j=1,n
**
**    many floating point operations: scope separation occured
XX    loop not optimized.
**
     do i=1,n
**
XX  Serial loop
**  --- not parallelised because increase in performance is not
          expected ---
     s=0.0d0
**
**    many floating point operations: scope separation occured
XX    loop not optimized.
**
     do k=1,n
**
```

演算量大=>scope　分割=>最適化,並列化が行われない。

並列化が行われないのは本当に演算量が多い事だけが原因かどうかは
Parallel Program Generator で確認できます。

```
 subroutine qdrmul10(aa,dd,bb,cc,n)
      implicit real*8 (a-h,o-z)
      data r/134217729.0d0/
      real*8  aa(5,n,n),dd(5,n),bb(5,n,n),cc(5,n,n)
      real*8  a(5),b(5),c(5),s(5)
C$OMP PARALLEL DO
C$OMP&  SCHEDULE(STATIC)
C$OMP&  PRIVATE(A)
C$OMP&  PRIVATE(B)
C$OMP&  PRIVATE(S)
                :
$OMP&  PRIVATE(P36)
C$OMP&  PRIVATE(P37)
      do j=1,n
      do i=1,n
      s=0.0d0
      do k=1,n
      end do
      cc(:,i,j)=s(:)
      end do
      end do
C$OMP END PARALLEL DO
      return
      end
```

Scopeに関して

以下の様な場合に使用します

1. コンパイル時間の伸びを抑える

演算量が非常に多い部分の最適化でコンパイル時間が非常に大きくなるのを防ぐため最適化の対象とする領域を分割するかどうかを定めます.
目安としては、opt(o(s))でのコンパイル時間は演算数の二乗に比例します。

2.乗加算命令が適用されるようにする

変数すべて倍精度

```
      T1=a*b          (1)
*soption split_scope
      T2=a*b-t1       (2)
```

最適化の範囲を(1)と(2)に分け,(2)で
乗加算命令が適用されa*bの値を誤差なく,
T1+T2の形で保持します。

scope (0) でコンパイル

```
 subroutine qdrmul10(aa,dd,bb,cc,n)
      implicit real*8 (a-h,o-z)
      data r/134217729.0d0/
      real*8  aa(5,n,n),dd(5,n),bb(5,n,n),cc(5,n,n)
      real*8  a(5),b(5),c(5),s(5)
**
**  Parallel processing starting at loop entry
**  Parallel function: _parallel_func_28_QDRMUL10
**  Parallel loop
      do j=1,n
**
**
**    IF test is invarient in loop so moved to outside.
**    Innermost loop resolved (2 times).
**    SWPL applied.
**
      do i=1,n
**
      s=0.0d0
**
**
**    Innermost loop resolved (2 times).
**    2 streams (BB[], DD[]) pre-fetch applied.
**    insufficient registers so compiler has generated
fixed load/store instruct
ions in the loop.
**
      do k=1,n
**
```

並列化されています。

2.2　10倍精度演算

2.2.1 行列積,QDR積計算

各精度行列積実行時間(秒)

精度	32core	64smp
6倍精度	1.877	1.304
8倍精度	3.285	2.140
10倍精度	7.58	5.275

10倍精度演算に特に問題は見られない。

10倍精度演算実行時間(秒)

演算	32core		64smp	
	scope(1)	scope(0)	scope(1)	scope(0)
行列積	6.964	6.996	4.823	4.745
QDR積	295.75	10.807	322.055	7.416

**QDR積計算は分割なしのソースで,6倍精度,
8倍精度演算と同じ方式のソース。
オプションはscope (1) が標準で
6倍精度,8倍精度演算ではscope (1) を使用。
Scope (0) は行列積演算には影響を与えない。**

2.2.2　量子モンテカルロ法による物性スペクトル計算

各精度実行時間(秒)		
精度	32core	64smp
4倍精度	9.424	11.250
6倍精度	28.837	23.743
8倍精度	51.789	42.074
10倍精度	109.902	90.533

（注）4倍精度は β＝10,u=5で実行

Smtの効果が4倍精度演算と多倍長精度演算で異なっている。

2.3 平方根計算アルゴリズム変更

$\beta=27, u=10$で正しく計算出来るために必要なビット数は251ビット。従来の逆数平方根を求める方式では$\beta=20, u=10$までしか正しく計算出来ていない。これは大きな数の逆数は必要とする精度のビットを持てない事による。
例えば10倍精度演算,$\beta=21, u=10$での平方根の計算では189ビット分の精度しか確保できないケースが発生している。
このため,平方根計算を除算を使用する方式に変更して、$\beta=27, u=10$まで10進10桁一致する答えを得た。

従来方式

$$x_0 = \frac{1}{\sqrt{a}}\text{(倍精度)}$$

$$x_{n+1} = x_n + \frac{x_n(1-ax_n^{\,2})}{2}$$

$$\sqrt{a} = a \times x_{last}$$

変更方式

$$x_0 = \sqrt{a}\text{(倍精度)}$$

$$x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$$

最終反復時は$x_{n+1} = \frac{1}{2}(x_n - \frac{a}{x_n}) + \frac{a}{x_n}$

$$\sqrt{a} = x_{last}$$

平方根計算アルゴリズム

実行時間(秒)

測定条件　$\beta = 20, u = 10$

方式	scope(1)		scope(0)	
	32core	64smp	32core	64smp
従来方式	648	712	110	91
変更方式	649	714	110	91

アルゴリズム変更は性能には影響していない。

3　BG/Qシステム

BG/Q 実行時間(秒)一覧表

精度	8smp	10smp	16smp	32smp	64smp
6倍精度	332	307	283	271	263
8倍精度	915	855	795	752	731
10倍精度(QDR分割なし)	2114	1960	1806	1684	1628
10倍精度(QDR分割あり)	2111	1958	1805	1689	1640

測定条件

1ノードで実行

平方根計算アルゴリズムは従来方式を使用

結果の特徴

1.10倍精度演算ではQDR積計算の分割あり，なしでの差はない。

2. Smt=4の性能が最も良いので,simd効果はないと見られる。

4. x5570とe5430システム

4.1　dd形式実行結果

x5570実行時間(秒)一覧表

精度	1smp	2smp	4smp	8smp
6倍精度	504	340	225	180
8倍精度	993	613	428	327
10倍精度(QDR分割なし)	2223	1407	966	779
10倍精度(QDR分割あり)	2314	1532	996	779

e5430実行時間(秒)一覧表

精度	1smp	2smp	4smp	8smp
6倍精度	741	437	301	235
8倍精度	1338	815	551	424
10倍精度(QDR分割なし)	2962	1810	1237	980
10倍精度(QDR分割あり)	3109	1889	1274	979

平方根計算アルゴリズムは従来方式を使用.

10倍精度演算ではQDR積計算の分割あり，なしでの差はない。

4.2 ieee754-2008形式実行結果

整数演算方式

シングルジョブ 実行時間(秒)
一覧表

測定条件

$L = 448 , N = 100 , \beta = 20 , u = 10$

精度	x5570	e5430
8倍精度	1457	2031
16倍精度	5029	6852
32倍精度	15073	20790

8ケース同時実行でのスループット測定

測定条件

$L = 448 , N = 100 , \beta = 20$

$u = 5.5 , 6.0 , ..., 8.5 , 9.0$

x5570	1547秒
e5430	2031秒

スループット測定結果

e5430 オーバーヘッドなし(測定誤差範囲内)

x5570 測定の環境設定の問題で
若干オーバーヘッドあり。

4.3　x5570　DQ形式実行結果

1. DD形式のソースを倍精度から4倍精度に変更したものをDQ形式とする。
2. 並列化はPPGENで作成したDD形式のソースをDQ形式に修正したものを使用。

x5570　DQ形式実行時間(秒)一覧表

(L=448,N=100)

精度	β	u	1smp	8smp
DQ	20	9	7200	2331
TQ	50	8	23852	8396
QQ	50	10	51622	18214
FQ	50	10	119412	43334

5. T2Kシステム

5.1　行列積,QDR積計算

行列積,QDR積16smp
実行時間(秒)一覧表

精度	実行時間
6倍精度行列積	8.048
8倍精度行列積	14.293
10倍精度行列積	33.139
10倍精度QDR積	48.576

5.2　DD形式実行結果

量子モンテカルロ法
物性スペクトル計算
実行時間(秒)一覧表
(16smp)

測定 1ノード
平方根計算アルゴリズム　従来方式
10倍精度演算　QDR分割なし

精度	実行時間
6倍精度	320
8倍精度	549
10倍精度	1256

5.3 ieee754-2008実行結果

整数演算方式8倍精度, DQ方式

$\beta = 20, u = 1.5, 2.0, \ldots\ldots, 8.5.9.0$　　フラット16mpi

整数演算方式8倍精度　2468秒

DQ　　14227秒