Vol.2014-ARC-213 No.28
Vol.2014-HPC-147 No.28
2014/12/10

XcalableMP による格子 QCD の並列化と Blue Gene/Q における性能評価

土井淳[†1]

ペタスケールからエクサスケールに向けての計算機の進化と共に，ますますの大規模な超並列分散化が必要となり，アプリケーション側での超並列化への対応が求められる．既存のアプリケーションの移植や新規のアプリケーション開発の生産性を高めつつ性能を引き出せるような開発環境が求められており，様々な言語などが提案されている．本報告ではその中から並列言語 XcalableMP を用いて，格子 QCD のソルバープログラムを並列化し，並列計算機 Blue Gene/Q を用いて性能を評価した結果を述べる．XcalableMP による並列化の手法は(i)グローバルビューモデルによる実装と，(ii)ローカルビューモデル（Co-array）による実装の 2 通りの実装方法があるが，それぞれを用いて格子 QCD を並列化し，数百ノードを超える並列度でのスケーラビリティを確認した．

Performance Evaluation of Lattice QCD Using XcalableMP on Blue Gene/Q

JUN DOI[†1]

As supercomputer evolves Petascale to Exascale, applications need much more massive distributed parallelization to utilize the capability and capacity of supercomputers. A lot of new parallel languages and development tools are presented recently to support porting existing applications or developing new applications for supercomputers and to enhance productivity and performance. In this report we selected parallel language XcalableMP to parallelize lattice QCD program to evaluate on the Blue Gene/Q supercomputer. There are 2 programming models in XcalableMP (i) Global view model and (ii) Local view model (Co-array), we implemented lattice QCD using both modes and evaluated scalability on more than 100 nodes on Blue Gene/Q.

1. はじめに

スーパーコンピューターのターゲットがエクサスケールコンピューティングへとシフトする中, 計算機のコア数や, 並列計算機の並列数がさらに増えていく傾向にある．もはや，高い並列度を活かすことがアプリケーション開発に必須になっていると言っても過言ではない．しかしながら，高い並列度を引き出すようなプログラミングを行うことは簡単なことではなく，時に専門的な知識や技術が必要になることもある．特にコンピュータサイエンスを本業としない科学者等の並列計算機ユーザにとっては，より簡単にプログラミングのできる環境が求められている．

そのような，高度な並列化をサポートするような新しい言語や開発環境等のツールは近年様々なものが提案されている．それぞれの言語やツール等には，それぞれ違った特性や長所や短所があり，必要に応じて使い分ける必要が出てくるが，それを知るためには経験的な知識の蓄積が必要となってくる．これらの新しい言語やツールにとって，ユーザによる使用経験事例や性能評価の数や種類を増やしていくことが重要である．

本報告では，そのような新しい並列言語のうち，XcalableMP[1]を選択した. XcalableMP は既存のプログラムに指示文を追加することで並列化を行うことができ，全く新しい言語を覚えるよりも手軽にプログラムの並列化を行うことができる．本報告では，格子 QCD プログラムを並列化し，その性能を評価した．

2. XcalableMP 概要

XcalableMP は，分散メモリ並列化を支援するための指示文をベースとした並列言語環境であり，既存の C 言語および Fortran 言語で書かれたプログラムに指示文を追加することで比較的簡単に分散メモリ並列化を行うことができる. また，OpenMP によるノード内の共有メモリ並列化を併用することができ，ハイブリッド並列化にも対応する. XcalableMP には，次の 2 つの使用方法がある.

(1)グローバルビューモデル

プログラムからは，配列全体と配列全体に対する処理が見えており，指示文を用いることで OpenMP でノード内の共有メモリ並列化を行うのと同じようにデータと処理の並列分散化を行うことができる. 分散されたデータに対して，袖領域を設定することができるため，ステンシル計算のような処理の並列化に向いている.

(2)ローカルビューモデル(Co-Array)

一般的に MPI を用いて並列化する場合のように各ノードがデータと処理を分散された状態でプログラミングを行う．そのため通信部分を明示的にプログラミングする必要

†1 日本アイ・ビー・エム株式会社 東京基礎研究所
IBM Research - Tokyo

Vol.2014-ARC-213 No.28
Vol.2014-HPC-147 No.28
2014/12/10

があるが，Co-array を用いることで，他のノードの持つ配列へ直接アクセスができるため，MPI より比較的簡単に記述ができ，また非同期的にデータの転送が行える．グローバルビューでは記述できないような通信パターンの場合でも Co-array を用いることで並列化が可能になる．

また，これら 2 つを組み合わせて使用することもでき，あるサブルーチンだけをローカルビューで記述するといったような書き方も可能である．

3. XcalableMP による格子 QCD の並列化

3.1 格子 QCD 概要

格子 QCD は，強い力の相互作用による場の理論を離散化し，コンピュータ上でシミュレーションできるようにしたものである．格子 QCD を用いることで様々な物理現象をコンピュータ上で再現することが可能であり，これまでに，カイラル対称性の自発的破れ[2]や，湯川理論における核力の再現[3]等がコンピュータシミュレーションによって再現されている．また，格子 QCD は古くからスーパーコンピュータの発展に寄与してきたアプリケーションの 1 つであるといっても過言ではなく，QCDPAX[4]，QCDSP[5]，QCDOC[6][7]，QPACE[8]等のように格子 QCD に特化したスーパーコンピュータも多く存在する．

格子 QCD のアルゴリズムには解こうとする問題や条件などによって様々な種類があるが，ほとんどのアルゴリズムに共通して計算コストの大きい部分は，CG 法などの反復法を用いて方程式を解く部分であり，その中でも特に Wilson-Dirac 演算子と呼ばれる計算が非常に時間のかかる処理である．本報告では，この CG 法を用いて方程式を解く部分についての並列化および性能評価を行う．

3.2 Wilson-Dirac 演算子の並列化

格子 QCD では 4 次元の時空間においてシミュレーションを行う．物理量は複素数で表現され，3 つの色を持った“カラー”，4 つのカラーを持つ“スピノル”にクォーク場の物理量が記述され，各 4 次元格子点上に定義される．また，クォークの相互作用に力を及ぼすグルーオン場が 4 次元格子間に定義され，それぞれ XYZT 方向毎に 3x3 の行列として定義される．Wilson-Dirac 演算子は，隣接格子のクォーク同士がグルーオン場の影響によって相互作用を受ける様子を計算するものであり，次の式で定義される．

$$D(n)=\delta(n) - \kappa\sum_{\mu=1}^{4}\left\{(1-\gamma_\mu)\cdot U_\mu(n)\cdot\delta(n+\hat{\mu})+(1+\gamma_\mu)\cdot U_\mu^t(n-\hat{\mu})\cdot\delta(n-\hat{\mu})\right\} \quad (1)$$

式(1)において，U はグルーオン場の行列，δ はクォーク場のスピノルをあらわす．μ は次元に対応し，γ は次に示すような 4x4 の複素行列である．

$$\gamma_1=\begin{pmatrix}0&0&0&-i\\0&0&-i&0\\0&i&0&0\\i&0&0&0\end{pmatrix}\quad \gamma_2=\begin{pmatrix}0&0&0&-1\\0&0&1&0\\0&1&0&0\\-1&0&0&0\end{pmatrix}$$

$$\gamma_3=\begin{pmatrix}0&0&-i&0\\0&0&0&i\\i&0&0&0\\0&-i&0&0\end{pmatrix}\quad \gamma_4=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&-1&0\\0&0&0&-1\end{pmatrix} \quad (2)$$

一般的に格子 QCD を並列化する場合は，4 次元格子のうち任意の軸方向の組み合わせについてブロック分割しそれぞれの軸方向にプロセスを割り当てる．小規模な並列化であれば T 軸方向のみに分割する方法が一般的であるが，大規模に並列化する場合は残りの軸方向についても分割していく．Wilson-Dirac 演算子では，隣接格子点についてのデータのみを参照するので，ブロック分割を行った際にも，隣接する格子点を持つノードとのみ通信を行えば良い．つまり，4 次元ブロック分割を行った場合は，隣接する 8 つのプロセスと隣接データを交換すれば良い．このとき注意すべきは，クォーク場は 4 次元格子点上に定義されるが，グルーオン場は 4 次元格子点間に定義されるという点である．グルーオン場は図 1 に示すように，ある格子点について見たときに，正方向の格子点間に定義する．図 1 では，ブロック分割を行った場合に，4x4 の格子をそれぞれのノードが持っている状態を示している．

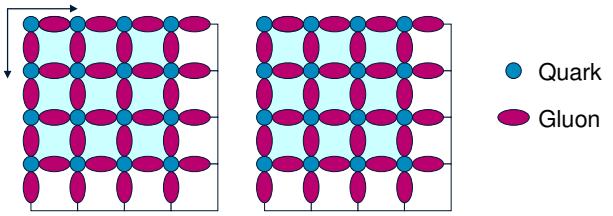

図 1 格子 QCD のブロック分割におけるクォーク場とグルーオン場の配置の例

グルーオン場をこのように配置するため，負の方向についての計算を行う場合，負の方向の端の格子点において隣接するグルーオン場は自分のノード内に存在しない．そのため負の方向の隣接ノードにおいて，あらかじめ隣接格子点のスピノルとグルーオンの行列の乗算を行ってからデータを転送する必要がある．なお，方程式を解いている間はグルーオン場は不変であるため，メモリ使用量が多くなるのとキャッシュの再利用性が落ちるのを気にしないのであれば，あらかじめ負方向に隣接するグルーオン場を転送しておき，重複して持っておく方法でも構わない．

ここで，式(2)のγ行列に着目すると，行列に対称性が見られることが分かる．この対称性を利用して，Wilson-Dirac 演算子において，本来 3x3 のグルーオン行列をスピノルの 4 つの 3 次元ベクトル（カラー）に乗じる必要があるとこ

Vol.2014-ARC-213 No.28
Vol.2014-HPC-147 No.28
2014/12/10

ろを，半分の 2 つのベクトルを乗じる処理に置き換えることができることが知られている．例えば次の式は，X の正方向についての乗算について，γ 行列によって共通項を求たものである．

$$(1-\gamma_1)\cdot U_1(n)\cdot\delta(n+\hat{\mu})=\begin{pmatrix} U_1(n)\cdot(s_1+i\cdot s_4) \\ U_1(n)\cdot(s_2+i\cdot s_3) \\ -i\cdot U_1(n)\cdot(s_2+i\cdot s_3) \\ -i\cdot U_1(n)\cdot(s_1+i\cdot s_4) \end{pmatrix} \quad (3)$$

この例では，$(s_1 + i\, s_4)$と$(s_2 + i\, s_3)$の 2 つについてグルーオン行列を乗じれば良いことが分かる．この 2 つについてここではハーフスピノルと呼ぶ．この性質を利用して，Wilson-Dirac 演算子は，次の 3 つのステップによって計算される．

(1) ハーフスピノルを計算する
(2) ハーフスピノルに 3x3 行列を乗じる
(3) ハーフスピノルを出力先のスピノルに加算する

また，隣接ノードとのデータ交換においても，この対称性を利用でき，通信量も半分にできる．正方向に隣接するプロセスにデータを送信する場合は相手のノードがこちら側の持つグルーオン行列を持っていないため，ステップ(2)までを行い行列を乗じた後のハーフスピノルを送信する．負方向に送信する場合は，ステップ(1)の後にハーフスピノルを送信すれば良い．

3.3 XcalableMP による Wilson-Dirac 演算子の並列化

3.3.1 グローバルビューによる並列化

クォーク場，グルーオン場共に，4 次元配列として扱い，4 次元ブロック分割を行うとして，次のようにテンプレートの定義を行う．

```
//Lattice size Lx*Ly*Lz*Lt, distributed in Px*Py*Pz*Pt
#pragma xmp nodes qcdNodes(Px,Py,Pz,Pt)
#pragma xmp template qcdTpl(0:Lx-1,0:Ly-1,0:Lz-1,0:Lt-1)
#pragma xmp distribute qcdTpl(block,block,block,block) onto qcdNodes
```

クォーク場およびグルーオン場には次のように袖領域は付けずに単純にブロック分割を行う．

```
//Gluon fields for each dimension
QCDMatrix Ux[LT][LZ][LY][LX];
QCDMatrix Uy[LT][LZ][LY][LX];
QCDMatrix Uz[LT][LZ][LY][LX];
QCDMatrix Ut[LT][LZ][LY][LX];
#pragma xmp align Ux[t][z][y][x] with qcdTpl(x,y,z,t)
#pragma xmp align Uy[t][z][y][x] with qcdTpl(x,y,z,t)
#pragma xmp align Uz[t][z][y][x] with qcdTpl(x,y,z,t)
#pragma xmp align Ut[t][z][y][x] with qcdTpl(x,y,z,t)

//Quark field example
QCDSpinor vx[LT][LZ][LY][LX];
#pragma xmp align vx[t][z][y][x] with qcdTpl(x,y,z,t)
```

ここで，QCDMatrix は 3x3 の複素ベクトルを持つ構造体（実際には 18 個の実数を持つ）で，QCDSpinor は 3x4 のスピノルを持つ構造体（実際には 24 個の実数を持つ）である．

ところで，Wilson-Dirac 演算子を並列化するにあたり，ノードの端の部分においてハーフスピノルを隣接ノード間で交換する必要があるが，グローバルビューによるプログラミングにおいて，端の部分だけ特別な処理をするということは非常に困難である．しかしながら，袖領域を持つ配列を用いれば，端の部分のデータ交換は非常に簡単に処理ができる．この機能を利用するため，前述の Wilson-Dirac 演算子における 3 ステップの計算について，ステップ(1)およびステップ(2)で計算されるハーフスピノルの値を保持するための袖領域つきの配列を，8 方向分（XYZT 軸の正負方向）用意し，それぞれの方向について計算結果を保存するようにする．こうすることで，特に端であることを意識せずに統一的な処理が行え，かつ簡単にデータ交換が行える．

このとき，この配列にデータを書き込むのは，正方向の処理の場合ステップ(1)の後，負方向の処理の場合ステップ(2)の後である．次のように 8 つの配列を定義する．袖領域はそれぞれの対応する方向にのみ付加し幅は 1 である．なお，QCDHalfSpinor は 3x2 のハーフスピノルを格納するための構造体（実際には 12 個の実数を持つ）である．

```
QCDHalfSpinor bufXP[LT][LZ][LY][LX];
QCDHalfSpinor bufXM[LT][LZ][LY][LX];
QCDHalfSpinor bufYP[LT][LZ][LY][LX];
QCDHalfSpinor bufYM[LT][LZ][LY][LX];
QCDHalfSpinor bufZP[LT][LZ][LY][LX];
QCDHalfSpinor bufZM[LT][LZ][LY][LX];
QCDHalfSpinor bufTP[LT][LZ][LY][LX];
QCDHalfSpinor bufTM[LT][LZ][LY][LX];
#pragma xmp align bufXP[t][z][y][x] with qcdTpl(x,y,z,t)
#pragma xmp align bufXM[t][z][y][x] with qcdTpl(x,y,z,t)
#pragma xmp align bufYP[t][z][y][x] with qcdTpl(x,y,z,t)
#pragma xmp align bufYM[t][z][y][x] with qcdTpl(x,y,z,t)
#pragma xmp align bufZP[t][z][y][x] with qcdTpl(x,y,z,t)
#pragma xmp align bufZM[t][z][y][x] with qcdTpl(x,y,z,t)
#pragma xmp align bufTP[t][z][y][x] with qcdTpl(x,y,z,t)
```

Vol.2014-ARC-213 No.28
Vol.2014-HPC-147 No.28
2014/12/10

```
#pragma xmp align bufTM[t][z][y][x] with qcdTpl(x,y,z,t)
#pragma xmp shadow bufXP[0][0][0][0:1]
#pragma xmp shadow bufXM[0][0][0][1:0]
#pragma xmp shadow bufYP[0][0][0:1][0]
#pragma xmp shadow bufYM[0][0][1:0][0]
#pragma xmp shadow bufZP[0][0:1][0][0]
#pragma xmp shadow bufZM[0][1:0][0][0]
#pragma xmp shadow bufTP[0:1][0][0][0]
#pragma xmp shadow bufTM[1:0][0][0][0]
```

また，次のようにデータの交換を同時に行う．async をつけることで異なる方向の通信同士を重ね合わせる．格子 QCD は周期的境界条件であるので，periodic を指定する．

```
#pragma xmp reflect (bufXP) width (0,0,0,/periodic/0:1) async(1)
#pragma xmp reflect (bufXM) width (0,0,0,/periodic/1:0) async(2)
#pragma xmp reflect (bufYP) width (0,0,/periodic/0:1,0) async(3)
#pragma xmp reflect (bufYM) width (0,0,/periodic/1:0,0) async(4)
#pragma xmp reflect (bufZP) width (0,/periodic/0:1,0,0) async(5)
#pragma xmp reflect (bufZM) width (0,/periodic/1:0,0,0) async(6)
#pragma xmp reflect (bufTP) width (/periodic/0:1,0,0,0) async(7)
#pragma xmp reflect (bufTM) width (/periodic/1:0,0,0,0) async(8)
#pragma xmp wait_async(1)
#pragma xmp wait_async(2)
#pragma xmp wait_async(3)
#pragma xmp wait_async(4)
#pragma xmp wait_async(5)
#pragma xmp wait_async(6)
#pragma xmp wait_async(7)
#pragma xmp wait_async(8)
```

グローバルビューによる Wilson-Dirac 演算子の処理は次のような 3 ステップになる．

(1) 正方向についてハーフスピノルを計算して保存，負方向についてハーフスピノルを計算しグルーオン行列を乗算
(2) 袖領域の交換
(3) 交換されたハーフスピノルを使って，正方向についてグルーオン行列を乗じて結果に加算，負方向について結果に加算する

3.3.2 Co-array による並列化

Co-array によって並列化を行う場合，MPI 等を用いて並列化を行う場合と同様に，それぞれのノードがブロック分割されたクォーク場およびグルーオン場を持つものとして処理を行う．グローバルビューによる並列化同様に，これらの配列は袖領域を持たないものとする．

隣接ノードとのデータ交換のためのハーフスピノルの配列について，グローバルビュー同様に全体を保持して袖領域を交換するようなプログラミングも可能ではあるが，ここではメモリアクセス量を減らすため，端の部分について送信するデータのみを保持するような配列を Co-array を用いて用意する．同様に受信するための配列も Co-array として定義する．次のような定義を行う．

```
#define MAX_L           32          //Maximum local lattice size
QCDHalfSpinor SendBuf[8][MAX_L*MAX_L*MAX_L];
QCDHalfSpinor RecvBuf[8][MAX_L*MAX_L*MAX_L];
#pragma xmp coarray SendBuf,RecvBuf:[*]
```

ところで，co-array を用いて隣接ノードとデータを交換する場合，ある方向に隣接するノードのノード番号を知る必要がある．4 次元のブロック分割を行った場合，次のようなマクロを用いて任意の 4 次元座標に対応するノードの番号を知ることができる．

```
#define GetRank(x,y,z,t) ¥
        ((x) + (y)*PX + (z)*PX*PY + (t)*PX*PY*PZ)
```

同様に自分のノード番号に対応する 4 次元座標は次のように求めることができる．

```
rank = xmp_node_num() - 1;
x = rank %PX;
y = (rank/PX) %PY;
z = (rank/(PX*PY)) %PZ;
t = rank/(PX*PY*PZ);
```

よって，X 正方向の隣接ノードの番号は次のように知ることができる．

```
rank_xp = GetRank((x+1) % PX,y,z,t);
```

これを利用して，co-array による隣接ノード間のデータ交換は次のように書ける．

```
//Local lattice size : Nx*Ny*Nz*Nt
//for X plus, send to X minus neighbor
RecvBuf[0][0:Ny*Nz*Nt]:[rank_xm+1] = SendBuf[0][0:Ny*Nz*Nt];
//for X minus, send to X plus neighbor
RecvBuf[1][0:Ny*Nz*Nt]:[rank_xp+1] = SendBuf[1][0:Ny*Nz*Nt];
//put other directions here
xmp_sync_all(st);
```

Vol.2014-ARC-213 No.28
Vol.2014-HPC-147 No.28
2014/12/10

実際の実装では，xmp_sync_all を行う前に，端以外の部分についての計算を行う．これは，グローバルビューによる実装のように端以外の部分についてはハーフスピノルを配列に保存しないため，別途計算をして直接結果に書き込んでしまう必要があるためである．また，ここにローカル計算を挿入することによって，通信と計算を非同期に同時に行うことで性能を向上させる狙いもある．

3.4 CG 法の並列化

ここでは次に示すような CG 法のコードを用意した．なお，U はグルーオン場の配列，D は Wilson-Dirac 演算子，それ以外の大文字のものはクォーク場の配列を表す．

```
S = B
X = B
R = B
sr = norm(S)
T = D(U,X)
S = D(U,T)
S = S* γ5
P = R = R - S
rrp = rr = norm(R)
do while(not converged)
        T = D(U,P)
        S = D(U,T)
        S = S* γ5
        pap = dot(S,P)
        cr = rr/pap
        X = cr*P + X
        R = -cr*S + R
        rr = norm(R)
        bk = rr/rrp
        P = bk*P + R
        rrp = rr
enddo
```

なお，γ_5は次のような行列である．

$$\gamma_5 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (4)$$

Wilson-Dirac 演算子については前述の通り並列化を行った．残りの処理はすべて線形代数であり，単純に各ノードにワークマッピングすることができる．ただし，norm および dot は全ノードで合計を求める必要があるため，reduction 指示文を使用して合計を求めた．

4. Blue Gene/Q における性能評価

4.1 Blue Gene/Q 概要

Blue Gene/Q[9]は第 3 世代目の Blue Gene スーパーコンピュータであり，ピーク性能 20Pflops をもつ米国ローレンスリバモア国立研究所に納入された Sequoia が代表的なシステムである．各計算ノードは 16 コアの組み込み PowerPC で，動作クロックは 1.6GHz である．また，FMA 演算をサポートした 4-way の SIMD 演算器があり，計算ノードあたり 204.8GFlops のピーク演算性能となる．それぞれの計算ノードに 16GB のメモリが搭載されており，計算ノード内で共有メモリ並列化（SMP）が利用でき，コアあたり 4 ハードウェアスレッド, 計算ノードあたり 64 スレッドまで利用できる．各計算ノードは 5 次元のトーラス通信網で相互接続されている．

4.2 Blue Gene/Q における XcalableMP

XcalableMP のバージョン 0.8 以降から Blue Gene/Q において動作するようになった．なお，本報告では，原稿執筆時においては，Nightly build version 20141001 を使用した．Blue Gene/Q では, グローバグビューにおける通信には MPI が，Co-array には GASNet[10]が使用される．どちらの通信ライブラリも共に PAMI と呼ばれる低レベル通信ライブラリが利用される．XcalableMP ではこれらを共存させるために，以下の 2 つの環境変数をセットする必要がある．

```
BG_MAPCOMMONHEAP=1
PAMI_CLIENTS=MPI,GASNet
```

4.3 格子 QCD の性能評価

本報告では，次の 3 つの実装について Blue Gene/Q の半筐体（512 計算ノード）を用いて性能を比較した．

(1) XcalableMP のグローバルビューを用いた実装
(2) XcalableMP の co-array を用いた実装
(3) MPI を用いた実装

MPI を用いた実装は，co-array を用いた実装のうち，端の部分のデータ交換を MPI_Isend および MPI_Irecv を用いて書き換え，reduction 指示文の部分を MPI_Allreduce に書き換えたのみである．

また，どの実装についても，OpenMP を利用してノード内並列化を行ったハイブリッド並列化の実装となる．

4.3.1 ハイブリッド並列化の評価

計算ノードあたりの XcalableMP のノード数（=MPI タスク数，以下 XMP ノード）と XMP ノードあたりのスレッド数を変えた場合の計算時間について評価する．ここで，計算ノードあたり合計で 32 スレッドを使用するような組み

Vol.2014-ARC-213 No.28
Vol.2014-HPC-147 No.28
2014/12/10

合わせについてそれぞれ計算ノードあたりの問題サイズが等しくなるようにして測定を行った．つまり，XMP ノード数が 2 の場合は XMP ノードあたり 16 スレッドとなり，それぞれの XMP ノードが XMP ノード数が 1 の場合の半分のサイズの問題を解く．Blue Gene/Q 512 計算ノードを使用し，計算ノードあたり 4096 格子点の問題についての測定値を Wilson-Dirac 演算子の処理時間について図 2 に，CG 法の処理時間について図 3 に示す．

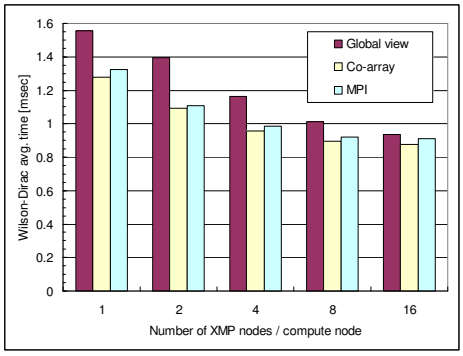

図 2 Blue Gene/Q 512 計算ノードにおける Wilson-Dirac 演算子のハイブリッド並列化の性能評価

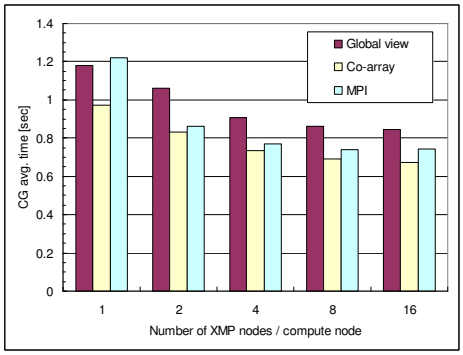

図 3 Blue Gene/Q 512 計算ノードにおける CG 法のハイブリッド並列化の性能評価

Wilson-Dirac 演算子が占める処理時間の割合が大きいので，CG 法についてもほとんど同じ傾向が見られ，計算ノードあたりの XMP ノード数を増やした方が性能が良くなった．これは，XcalableMP の指示文の制約により，OpenMP の parallel 指示文を XcalableMP の指示文より外に定義できないため，OpenMP のオーバーヘッドが大きく出ているためと考えられる．以降の性能評価について，計算ノードあたり 16XMP ノードで測定を行った．

また全体的に，グローバルビューによる実装よりも，Co-array による実装の方が性能が良く，これは，通信と計算の重ね合わせによるものと，ハーフスピノルの配列へのアクセス量の違いから来るものと考えられる．また，Co-array による実装の方が MPI による実装より若干速く，これは GASNet による非同期通信が MPI の非同期通信よりも良い性能が出ているものと思われる．

4.3.2 スケーラビリティの評価

Blue Gene/Q 512 計算ノードまでを用いて，強スケーリングおよび弱スケーリングについて評価する．

まず，全体の格子サイズを，16x16x16x32 と 32x32x32x64 の 2 通りに固定し，使用する計算ノード数を変えたときの性能を見る強スケーリングについて，図 4 と図 5 に Wilson-Dirac 演算子の測定結果を，図 6 と図 7 に CG 法の測定結果をまとめる．

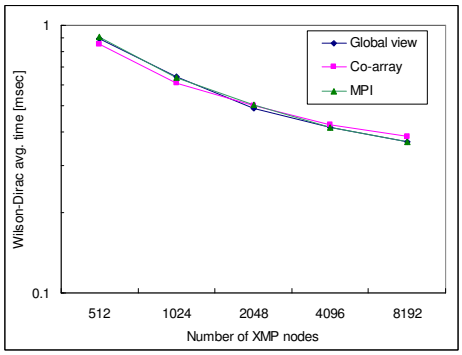

図 4 Blue Gene/Q における Wilson-Dirac 演算子の強スケーリング（格子サイズ 16x16x16x32）

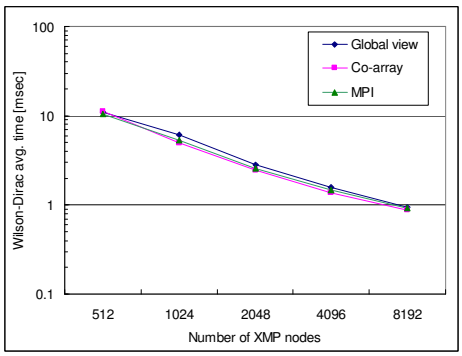

図 5 Blue Gene/Q における Wilson-Dirac 演算子の強スケーリング（格子サイズ 32x32x32x64）

Wilson-Dirac 演算子については，いずれの実装方法を用いてもおおよそ同じような結果になった．格子サイズが比較的小さい 16x16x16x32 の場合使用する XMP ノード数を増やすと XMP ノードあたりに処理する量が減ってスケーラビリティが悪化するのが確認できたが，32x32x32x64 の方では良好なスケーラビリティが得られた．

Vol.2014-ARC-213 No.28
Vol.2014-HPC-147 No.28
2014/12/10

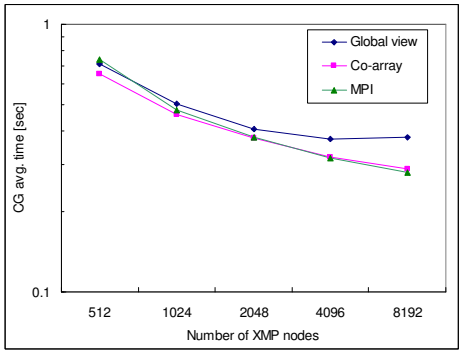

図 6 Blue Gene/Q における CG 法の強スケーリング（格子サイズ 16x16x16x32）

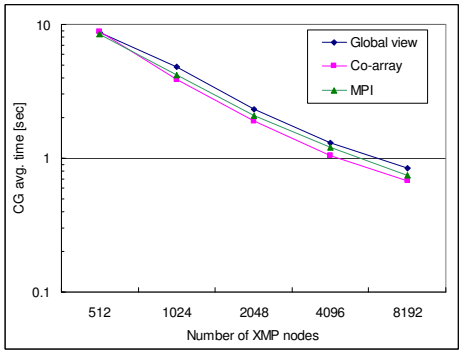

図 7 Blue Gene/Q における CG 法の強スケーリング（格子サイズ 32x32x32x64）

CG 法については，比較的サイズの小さい 16x16x16x32 において，グローバルビューによる実装で性能の劣化が確認できた．

次に，計算ノードあたりの格子サイズを 256 と 1024 の 2 通りに固定し，使用する計算ノード数を変えたときの性能を見る弱スケーリングについて，図 8 と図 9 に Wilson-Dirac 演算子の測定結果を，図 10 と図 11 に CG 法の測定結果をまとめる．

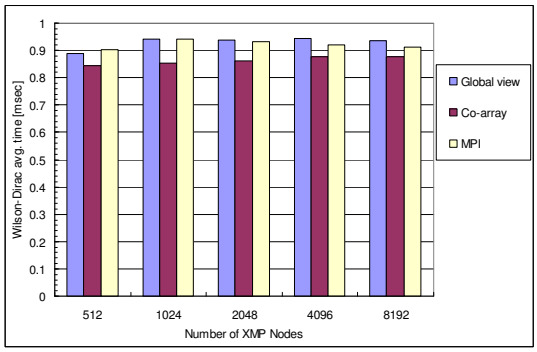

図 8 Blue Gene/Q における Wilson-Dirac 演算子の弱スケーリング（計算ノードあたり格子サイズ 256）

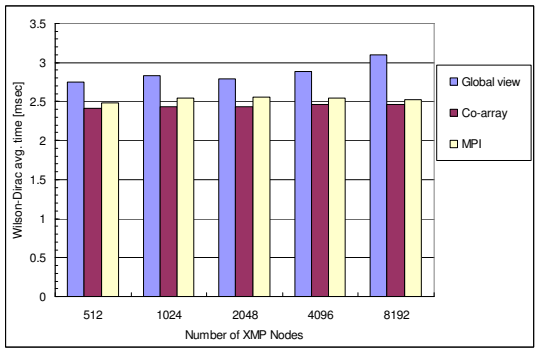

図 9 Blue Gene/Q における Wilson-Dirac 演算子の弱スケーリング（計算ノードあたり格子サイズ 1024）

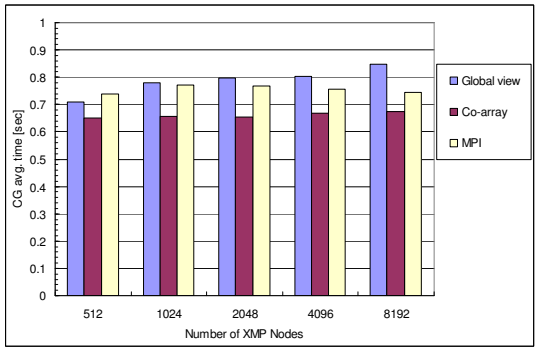

図 10 Blue Gene/Q における CG 法の弱スケーリング（計算ノードあたり格子サイズ 256）

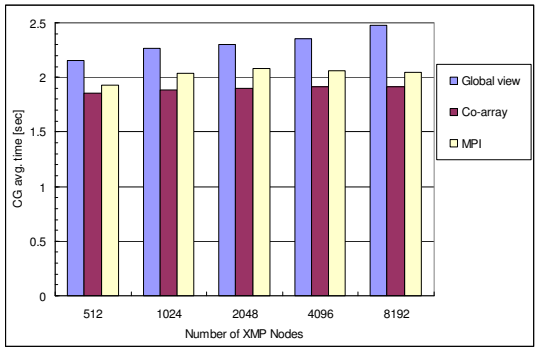

図 11 Blue Gene/Q における CG 法の弱スケーリング（計算ノードあたり格子サイズ 1024）

いずれの場合も，ほぼ同じような傾向が見られ，グローバルビューによる実装以外では，非常に良好なスケーラビリティが確認できた．グローバルビューによる実装では，ハーフスピノルを一旦すべての格子点について保存する方法をとっているため，そこがボトルネックになっている可能性がある．

Vol.2014-ARC-213 No.28
Vol.2014-HPC-147 No.28
2014/12/10

5. おわりに

ここまでの性能測定の結果，XcalableMP を用いて格子 QCD を実装したところ，おおむね MPI による実装と比べて遜色ない性能が得られることが分かった．グローバルビューによる実装においては，若干性能が劣るもののシリアルプログラムに指示文を追加しただけにしては十分すぎる性能であると言える．また，Co-array を用いた実装では，プログラミング手法も MPI とほぼ変わらずに実装できる上に，MPI よりも若干良い性能が得られた．

なお，本報告では，単純な実装のみの評価を行い，突き詰めた最適化までは行っていない．今後は，XcalableMP を使った最適化にも着目し，格子 QCD に限らず他のアプリケーションについても実装を行いたい．また，X10[11]などの他の並列プログラミング言語との比較や，Blue Gene/Q 以外の並列計算機における評価も行いたい．

謝辞 本報告にて使用した格子 QCD のプログラムは，松古栄夫氏の“Lattice QCD in C/C++ for tuning test bed” (http://research.kek.jp/people/matufuru/Research/Programs/Tuning_Cpp/)を元に作成した．

参考文献

1) XcalableMP WebSite, http://xcalablemp.org/
2) H. Fukaya et al. [JLQCD collaboration], Two-flavor lattice QCD simulation in the epsilon-regime with exact chiral symmetry, Physical Review Letters 98, 172001, 2007.
3) N. Ishii et al. Nuclear force from lattice QCD, Physical Review Letters, June, 2007.
4) T. Shirakawa et al. QCDPAX–an MIMD array vector processors for the numerical simulation of quantum chromodynamics, Proceedings of the 1989 ACM/IEEE conference on Supercomputing, 1989.
5) R. D. Mawhinney, The 1 Teraflops QCDSP Computer, Parallel Computer 25, No. 10/11, pp.1281-1296, September, 1999.
6) P. A. Boyle et al. QCDOC: A 10–Teraflops Computer for Tightly-Coupled Calculations, Proceedings of the ACM/IEEE conference on Supercomputing SC04, 2004.
7) P. A. Boyle et al. Overview of the QCDSP and QCDOC computers, IBM J. of Research and Development Vol. 49, No.2/3, pp.351-365, 2005.
8) H. Baier et al. QPACE – a QCD parallel computer based on Cell processors, PoS 27th International Symposium on Lattice Field Theory, Lattice2009, 2009.
9) IBM Blue Gene/Q, http://www-03.ibm.com/systems/technicalcomputing/solutions/bluegene/
10) GASNet Communication System, http://gasnet.lbl.gov/
11) X10: Performance and Productivity at Scale, http://x10-lang.org/