富士通 PRIMEHPC FX10 チューニング連載講座 IV

疎行列演算の高速化：OpenMP によるノード内並列化（その 2）

中 島 研 吾

東京大学情報基盤センター

（第 1 回（5 月号）より続く）

6. 計算結果

前号において，MC 法（Multicoloring），CM 法（Cuthill-McKee），RCM 法（Reverse Cuthill-McKee）等による並び替えによって ICCG 法の IC 前処理部分のデータ依存性を排除し，並列性を抽出することが可能となることを示した。図 14～16 に，NX=NY=NZ=100（10^6 要素）の場合の，Oakleaf-FX　1 ノード，16 コア（16 スレッド）における計算結果を示す。横軸は色数で，RCM 法の場合はレベル数に相当する。

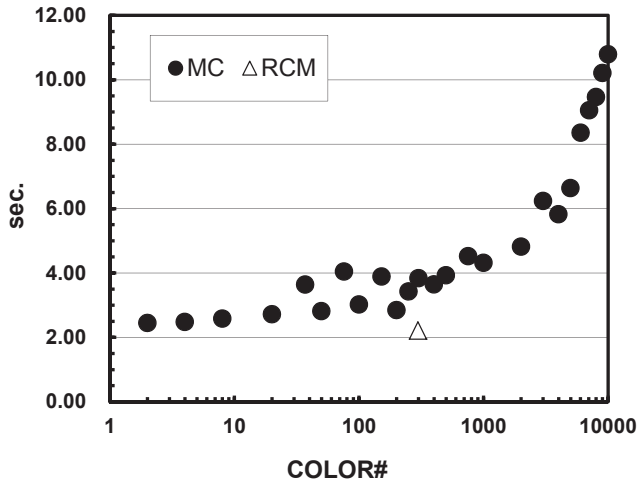

図 14　並列 ICCG 法計算結果（ICCG 法計算時間）：10^6 要素，Oakleaf-FX　1 ノード 16 コア

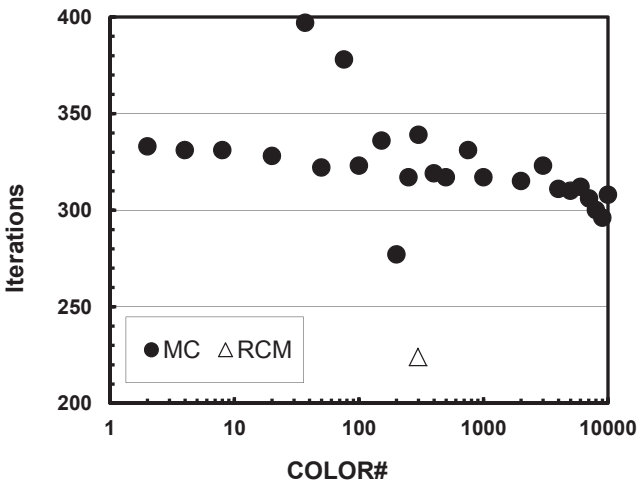

図 15　並列 ICCG 法計算結果（ICCG 法反復回数）：10^6 要素，Oakleaf-FX　1 ノード 16 コア

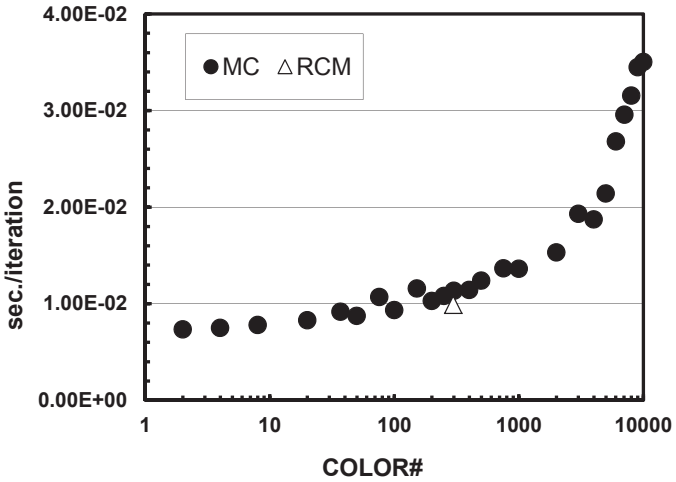

図 16　並列 ICCG 法計算結果（ICCG 法反復あたり計算時間）：
10^6 要素，Oakleaf-FX　1 ノード 16 コア

MC は色数が増加するとともに，反復回数は概ね減少するが（図 15），OpenMP のオーバーヘッドのため一反復あたりのコストが増加し（図 16），結果として計算時間は色数が増加するほど長くなる（図 14）。RCM 法（レベル数 298）の反復回数は 224 回と MC 法（約 300 回～330 回）と比較して少ないが（図 15），色数 200～300 程度の MC 法の場合と同じ程度の計算コストである（図 16）。結果的には図 14 に示すように，RCM 法の計算時間（2.21 秒）は MC 法の最適なケース（2 色，2.44 秒）よりも短い。

MC法の色数のICCG法の収束性への効果については，これまで様々な研究によって説明が試みられているが，〔1〕においては土肥らによって「Incompatible Nodes（以下ICN）」の概念に基づいて説明されている。図17～20に示した16要素の二次元体系において，節点番号順に前進代入あるいはGauss-Seidelのような操作を施した場合，節点1以外の節点は全て，番号の若い要素の影響を受ける。ICNとは，この図における要素1，すなわち，他の要素から影響を受けない要素のことである（図17）。文献〔1〕によれば，一般にICNの数が少ないほど，他の要素の計算結果の効果を考慮しながら計算が実施されていることから，収束が良い。

2色に塗り分けるred-black ordering の場合，多くのICNを持つ（図18）。また，色数を増やして，4色にすると，図19に示すように，ICNの数は減少する。基本的に色数を増加させるとICNの数は減少する（非常に複雑な形状の場合などで例外はあるが）。また，CM法の場合は，図20に示すようにICNの数は1である。MC法では，各色における要素の独立性のみが考慮されているのに対してCM法，RCM法では，各レベル（色）における要素の独立性とともに，各レベル（色）間の依存性についても考慮されており，不完全修正コレスキー分解，前進後退代入における計算順序に適合した並び替えとなっている。図21は色数とIncompatible Node数の関係を示す。

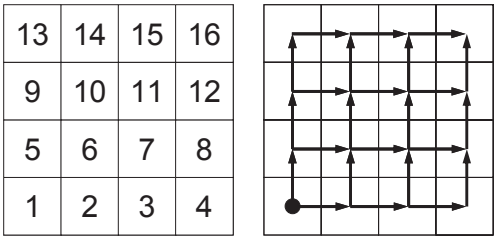

図17　初期状態（●：Incompatible Nodes）

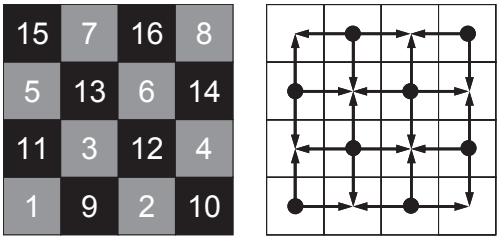

図18　MC（色数:2）（●：Incompatible Nodes）

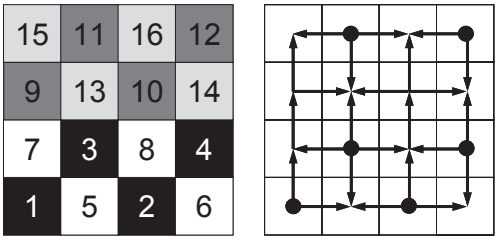

図19　MC（色数:4）（●：Incompatible Nodes）

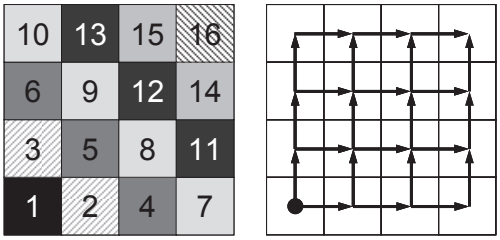

図20　CM（レベル数:7）（●：Incompatible Nodes）

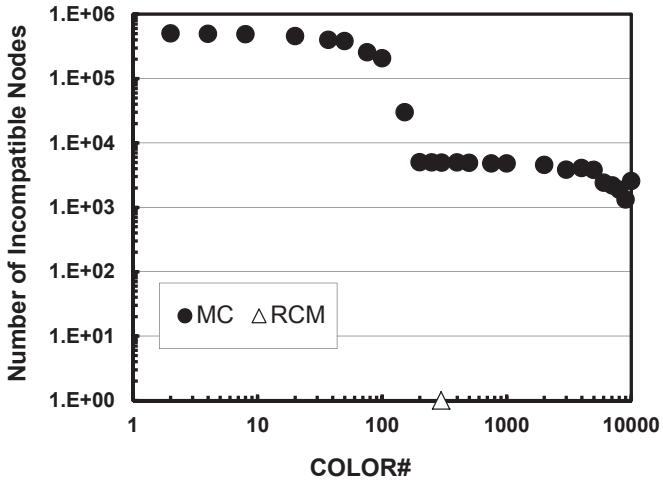

図 21　Incompatible Node（ICN）の数（10^6 要素）

7. 性能向上の方法

(1) 概要

前章までに，有限体積法によって離散化された三次元領域で，ポアソン方程式を ICCG 法によって解くプログラムを OpenMP によって並列化する方法について紹介した。ここでは，更なる性能向上の方法について紹介する。

本稿では以下に示すように，段階的に最適化を実施する。

- ケース 0：初期状態
- ケース 1：Sequential Reordering の適用
- ケース 2：Sequential Reordering +ELL フォーマットの適用

(2) ケース 1 (Sequential Reordering の適用)

現状の手法では図 13 に示すように：

① 同一の色（またはレベル）に属する要素は独立であり，並列に計算可能
② 「色」の順番に番号付け
③ 色内の要素を各スレッドに振り分ける

という方式を採用しているが，同じスレッド（すなわち同じコア）に属する要素は連続の番号では無いため，特に色数が増加すると性能が低下する可能性がある。すなわち，色数が多くなると次の色の処理に移る際にアクセスするデータのアドレスが大きく移動し，プリフェッチしたデータが利用されず，新たにアクセスするデータはプリフェッチされていないためにデマンドアクセスが発生する。それに対して，同じスレッドに属する要素を連続な番号付けになるようにすると，アクセスの連続性が高まり，特に色数が多い場合の性能向上が期待できる。

ケース 1 では同じスレッドで処理するデータをなるべく連続に配置するように更に並び替え（sequential reordering），効率の向上を図ることとした。番号の付け替えによって要素番号の大小関係は変わる可能性があるが，上三角，下三角の関係は変えず，元の計算と反復回数が変わらないようにする．従って自分より要素番号が大きいのに下三角成分に含まれているような場合もありうる。Sequential reordering は first-touch data placement と組み合わせることにより，T2K 東大における最適化手法としても有効な手法である〔2,3〕。Sequential reordering（ordering）に対して元の並べ方を coalesced ordering と呼ぶ。

図 22 はこのよう手法を実現するための配列「SMPindex_new」の考え方であり，リスト 15 は「SMPindex⇒SMPindex_new」の並べ替えの実装である。リスト 16，17 はそれぞれ，前進代入，行列ベクトル積を「SMPindex_new」を使って実装した場合の適用例である。配列「SMPindexG」は連続な番号付けで無いため，リスト 17 に示したように，行列ベクトル積においても「SMPindex_new」を使わなければならない。

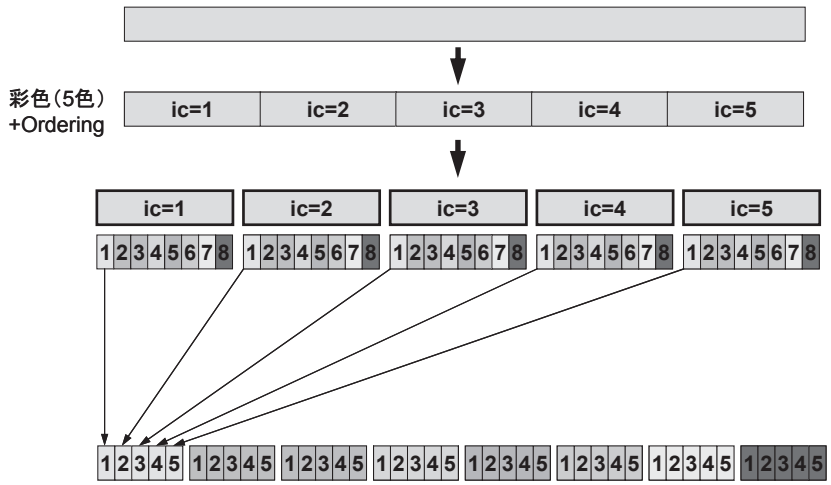

図 22　Sequential Reordering における配列「SMPindex_new」の考え方：
各スレッド上の要素は連続した番号付け

```
      allocate (SMPindex(0:PEsmpTOT*NCOLORtot))
      SMPindex= 0
      do ic= 1, NCOLORtot
        nn1= COLORindex(ic) - COLORindex(ic-1)
        num= nn1 / PEsmpTOT
        nr = nn1 - PEsmpTOT*num
        do ip= 1, PEsmpTOT
          if (ip.le.nr) then
            SMPindex((ic-1)*PEsmpTOT+ip)= num + 1
          else
            SMPindex((ic-1)*PEsmpTOT+ip)= num
          endif
      enddo; enddo

      allocate (SMPindex_new(0:PEsmpTOT*NCOLORtot))
      SMPindex_new(0)= 0
      do ic= 1, NCOLORtot
        do ip= 1, PEsmpTOT
          j1= (ic-1)*PEsmpTOT + ip
          j0= j1 - 1
          SMPindex_new((ip-1)*NCOLORtot+ic)= SMPindex(j1)
          SMPindex(j1)= SMPindex(j0) + SMPindex(j1)
      enddo; enddo

      do ip= 1, PEsmpTOT
        do ic= 1, NCOLORtot
          j1= (ip-1)*NCOLORtot + ic
          j0= j1 - 1
          SMPindex_new(j1)= SMPindex_new(j0) + SMPindex_new(j1)
      enddo; enddo
```

リスト 15　「SMPindex⇒SMPindex_new」の変換

```
ORIGINAL
!$omp parallel private(ip,ip1,i,WVAL,j)
      do ic= 1, NCOLORtot
!$omp do
        do ip= 1, PEsmpTOT
          ip1= (ic-1)*PEsmpTOT + ip
          do i= SMPindex(ip1-1)+1, SMPindex(ip1)
            WVAL= W(i,Z)
            do k= indexL(i-1)+1, indexL(i)
              WVAL= WVAL -  AL(k) * W(itemL(k),Z)
            enddo
            W(i,Z)= WVAL * W(i,DD)
        enddo; enddo
      enddo
!$omp end parallel


NEW
!$omp parallel private(ip,ip1,i,WVAL,j)
      do ic= 1, NCOLORtot
!$omp do
        do ip= 1, PEsmpTOT
          ip1= (ip-1)*NCOLORtot + ic
          do i= SMPindex_new(ip1-1)+1, SMPindex_new(ip1)
            WVAL= W(i,Z)
            do k= indexL(i-1)+1, indexL(i)
              WVAL= WVAL -  AL(k) * W(itemL(k),Z)
            enddo
            W(i,Z)= WVAL * W(i,DD)
        enddo; enddo
      enddo
!$omp end parallel
```

リスト 16　前進代入（[M]{z}={r}）のループの OpenMP による並列化（新旧の比較）

```
ORIGINAL
!$omp parallel do private(ip,i,VAL,j)
      do ip= 1, PEsmpTOT
        do i = SMPindexG(ip-1)+1, SMPindexG(ip)
          VAL= D(i)*W(i,P)
          do j= 1, INL(i)
            VAL= VAL + AL(j,i)*W(IAL(j,i),P)
          enddo
          do j= 1, INU(i)
            VAL= VAL + AU(j,i)*W(IAU(j,i),P)
          enddo
          W(i,Q)= VAL
        enddo
      enddo
!$omp end parallel do

NEW
!$omp parallel do private(ip,i,VAL,j)
      do ip= 1, PEsmpTOT
        do i= SMPindex_new((ip-1)*NCOLORtot)+1, SMPindex_new(ip*NCOLORtot)
          VAL= D(i)*W(i,P)
          do j= 1, INL(i)
            VAL= VAL + AL(j,i)*W(IAL(j,i),P)
          enddo
          do j= 1, INU(i)
            VAL= VAL + AU(j,i)*W(IAU(j,i),P)
          enddo
          W(i,Q)= VAL
        enddo
      enddo
!$omp end parallel do
```

リスト 17　行列ベクトル積（{q}=[A]{p}）のループの OpenMP による並列化（新旧の比較）

(3) ケース 2 (Sequential Reordering+ELL フォーマットの適用)

これまでに紹介したプログラムでは，疎行列の格納法として CRS（Compressed Row Storage）を採用している。ここでは前節で紹介した sequential reordering に加えて，更に行列格納手法に変更を加えた場合について検討する。

CRS は疎行列格納法として広く使用されており，不規則な疎行列を効率よく記憶できる。ELL（Ellpack-Itpack）フォーマットは図 23（b）に示すように，各行あたりの成分数を最大値に固定し，実際に値のないところには 0 を入れる手法である。疎行列計算は memory-bound なプロセスであるが，ELL フォーマットの採用により，最内ループ長が固定されるため，キャッシュが有効利用できるようになり，メモリアクセス性能も高まるものと期待される。本稿では RCM の場合にのみ ELL フォーマットを適用した。

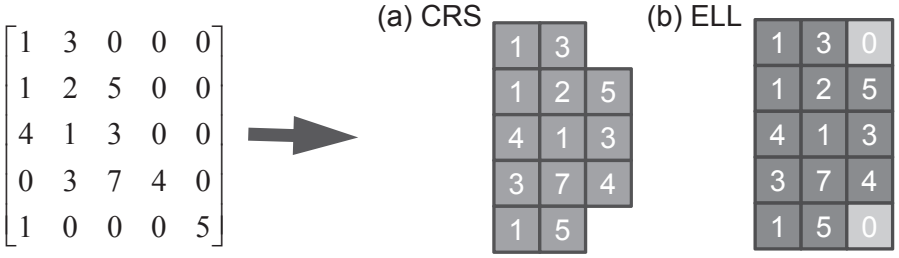

図 23　疎行列の格納手法（a）CRS（Compressed Row Storage），（b）ELL（Ellpack-Itpack）

(4) 計算結果

表 3 に NX=NY=NZ=100（10^6 要素）の場合の，Oakleaf-FX　1 ノード，16 コア（16 スレッド）における計算結果を示す。MC（2 色）の場合は sequential reordering の効果はほとんどなく，むしろ遅くなっているが，MC（200 色），RCM（298 レベル）の場合は 5〜7%程度の性能向上が見られる。ELL の効果は顕著であり，sequential reordering と組み合わせることにより，ケース 0 のときと比べて 60%程度の性能向上が得られた。

表 3　最適化の効果：ICCG 法の計算時間（単位：秒），10^6 要素，Oakleaf-FX　1 ノード 16 コア

	反復回数	ケース 0	ケース 1（sequential reordering）	ケース 2（sequential reordering + ELL）
MC（2 色）	333	2.44	2.47	―
MC（200 色）	277	2.85	2.71	―
RCM（298 レベル）	224	2.21	2.07	1.37

表 4 は NX=NY=NZ=128（2,097,152 要素）の場合の，Oakleaf-FX　1 ノード，16 コア（16 スレッド）における前進後退代入部分の計算を FX10 の詳細プロファイラの精密 PA 可視化機能[1]によって評価した結果である。Sequential reordering，ELL フォーマットの適用によって計算性能が向上していることがよくわかる。特にメモリースループットがケース 0 からケース 2 で 2 倍近く向上している。

表 3　最適化の効果：2,097,152 要素，RCM（382 レベル），Oakleaf-FX　1 ノード 16 コア

	ケース 0	ケース 1（sequential reordering）	ケース 2（sequential reordering + ELL）
ICCG 法計算時間（秒）	6.37	5.78	3.61
前進後退代入部分 L1 デマンドミス率（%）	37.7	29.3	16.5
前進後退代入部分 メモリースループット（GB/sec.）	28.7	32.6	52.2

8. おわりに

有限要素法，有限体積法などの疎行列を係数行列とし，間接参照を伴うアプリケーションについて，Oakleaf-FX　1 ノードにおいて OpenMP を使用して並列化した事例について説明した。互いに独立な要素を抽出して，彩色，再番号づけすることによって，ICCG 法の修正不完全コレスキー分解，前進後退代入などの処理において，データ依存性を除去することが可能である。更に sequential reordering，ELL フォーマットの適用によって性能を高めることができる。

今回は 1 ノードにおける並列化事例を紹介したが，本稿で紹介した考え方はマルチノードにおける OpenMP／MPI ハイブリッド並列プログラミングモデルに容易に拡張可能である〔4〕。

[1] https://oakleaf-www.cc.u-tokyo.ac.jp/cgi-bin/hpcportal.ja/index.cgi

このような事例についてはまた改めて別稿にて紹介する予定である。

参 考 文 献

[1] Doi, S. and Washio, T. (1999) Using Multicolor Ordering with Many Colors to Strike a Better Balance between Parallelism and Convergence, RIKEN Symposium on Linear Algebra and its Applications, 19-26

[2] 中島研吾：T2K オープンスパコン（東大）チューニング連載講座（その 5）OpenMP による並列化のテクニック : Hybrid 並列化に向けて，スーパーコンピューティングニュース（東京大学情報基盤センター）11-1（2009）
http://www.cc.u-tokyo.ac.jp/support/press/news/VOL11/No1/200901tuning.pdf

[3] Nakajima, K.: Flat MPI vs. Hybrid: Evaluation of Parallel Programming Models for Preconditioned Iterative Solvers on “T2K Open Supercomputer”, IEEE Proceedings of the 38th International Conference on Parallel Processing (ICPP-09), pp.73-80 (2009)

[4] Nakajima, K.: Large-scale Simulations of 3D Groundwater Flow using Parallel Geometric Multigrid Method, Procedia Computer Science 18, pp.1265-1274 (2013)