

# 拡張型 ELL 行列格納手法に基づくメニコア向け疎行列ソルバー

中島研吾<sup>†1 †2</sup>

ELL 格納手法は行列ベクトル積の最適化に広く使用されている。本研究では、ELL 格納手法をメニコア向けにスレッド並列化された ICCG 法ソルバー向けに拡張し、Intel Xeon Phi, Fujitsu FX10, Intel Ivy-Bridge 上で性能評価を実施した。

## Sparse linear solver based on extended ELL storage format of coefficient matrix for manycore architectures

KENGO NAKAJIMA<sup>†1 †2</sup>

ELL storage format is one of the optimum methods for SpMV operations. In the present work, extended version of ELL storage format is proposed for multithreaded ICCG solver on manycore architectures. Proposed method has been evaluated on Intel Xeon Phi, Fujitsu FX10, and Intel Ivy-Bridge.

### 1. はじめに

本研究では、有限体積法によるポアソン方程式ソルバー [1,2] から導かれる対称正定な疎行列を係数とする連立一次方程式を不完全コレスキー分解前処理付き共役勾配法 (Preconditioned Conjugate Gradient Method by Incomplete Cholesky Factorization, ICCG 法) によってメニコアクラスで効率よく解くための手法を検討する。本研究では行列格納手法、特に ELLpack-Itpack (ELL) 形式とその拡張に着目する。OpenMP/MPI ハイブリッド並列プログラミングモデルを使用することを想定し、計算ノード上において OpenMP を使用してスレッド並列化したプログラムを対象とする。本研究で検討した手法を Intel Xeon Phi, Fujitsu PRIMEHPC FX10 (Fujitsu FX10) [3], Intel Sandy Bridge にて評価した。

本稿では以下、対象とするアプリケーション、使用した計算機の概要、最適化手法、計算結果について紹介する。

### 2. 対象とするアプリケーション

本稿で対象とするアプリケーションは図 1 に示す差分格子によってメッシュ分割された三次元領域において、以下のポアソン方程式を解くものである [1] :

$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = f \quad (1)$$

$$\phi = 0 @ z = z_{\max} \quad (2)$$

形状は規則正しい差分格子であるが、プログラムの中では、一般性を持たせるために、有限体積法に基づき、非構

造格子型のデータとして考慮する。

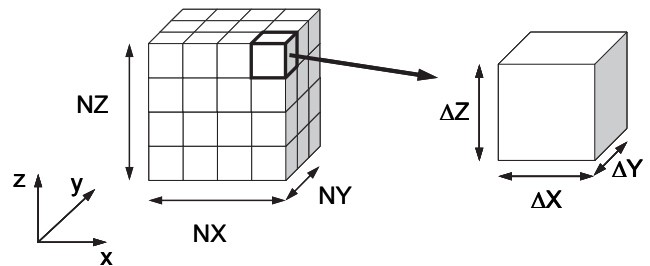


図 1 三次元ポアソン方程式ソルバーの解析対象差分格子の各メッシュは直方体 (辺長さは  $\Delta X, \Delta Y, \Delta Z$ ),  $X, Y, Z$  各方向のメッシュ数は  $NX, NY, NZ$

図 1 における任意のメッシュ  $i$  の各面 (6 面) を通過するフラックスについて、式 (1) により以下に示す式 (3) が得られる :

$$\left[ \sum_{k=1}^6 \frac{S_{ik}}{d_{ik}} \right] \phi_i - \left[ \sum_{k=1}^6 \frac{S_{ik}}{d_{ik}} \phi_k \right] = +V_i f_i \quad (3)$$

ここで、 $S_{ik}$ : メッシュ  $i$  と隣接メッシュ  $k$  間の表面積、 $d_{ik}$ : メッシュ  $i-k$  重心間の距離、 $V_i$ : メッシュ  $i$  の体積、 $f_i$ : メッシュ  $i$  の体積あたりフラックスである。これは各メッシュ  $i$  について成立する式であり、全メッシュ数を  $N$  とすると、 $N$  個の方程式を連立させて、境界条件を適用し、連立一次方程式  $[A]\{\phi\} = \{b\}$  を解くことで解を得る。式 (3) の左辺第一項は  $[A]$  の対角項、第二項は非対角項、右辺は  $\{b\}$  に対応する。各メッシュ  $i$  に対応する非対角成分数は最大 6 個であるので、係数行列  $[A]$  は疎 (sparse) な行列となる。

係数行列  $[A]$  は対称かつ正定 (Symmetric Positive Definite, SPD) であるため、前処理付き共役勾配法 (Preconditioned Conjugate Gradient Method) を適用する。前処理手法としては、対称行列向けに広く使用されている不完全コレスキー分解 (Incomplete Cholesky Factorization, IC) を使用する [1,2].

<sup>†1</sup> 東京大学情報基盤センター  
 Information Technology Center, The University of Tokyo  
<sup>†2</sup> 科学技術振興機構 CREST  
 CREST, Japan Science and Technology Agency

本研究では、係数行列は対称であるが、プログラム内では上下三角成分を別々に記憶している [1]。本研究では、fill-in を考慮しない IC(0) を使用している。

不完全コレスキー分解を前処理手法とする共役勾配法を ICCG 法と呼ぶ。ICCG 法では、不完全コレスキー分解生成時、前進代入、後退代入でメモリへの書き込みと参照が同時に生じ、データ依存性が発生する可能性があるため、リオーダーリングが必要である [1,2]。

### 3. 計算機環境

本研究では以下の 3 種類の計算機環境を使用した：

- **FX10** : Fujitsu PRIMEHPC FX10 に基づく東京大学情報基盤センターの Oakleaf-FX システム [3]
- **MIC** : Intel Xeon Phi (Knights Corner)
- **IvyB** : Intel Xeon E5 (IvyBridge-EP)

プログラムは Fortran90 で記述しており、FX10 では富士通製コンパイラ、MIC、IvyB では Intel Comliler (Ver.15) / Intel Parallel Studio XE 2015 を使用した。表 1 に計算機環境の概要を示す。

表 1 各計算環境 (1 ソケット) の概要

略 称	FX10	MIC	IvyB
名 称	Fujitsu SPARC64 IX fx	Intel Xeon Phi 5110P (Knights Corner)	Intel Xeon E5-2680 v2 (Ivy-Bridge-EP)
動作周波数 (GHz)	1.848	1.053	2.80
コア数 (有効スレッド数)	16 (16)	60 (240)	10 (20)
使用スレッド数	16	240	10
メモリ種別	DDR3	GDDR5	DDR3
理論演算性能 (GFLOPS)	236.5	1,010.9	224.0
主記憶容量 (GB)	32	8	64
理論メモリ性能 (GB/sec.)	85.1	320	59.7
キャッシュ構成	L1:32KB/core L2:12MB/socket	L1:32KB/core L2:512KB/core	L1:32KB/core L2:256KB/core L3:25MB/socket
コンパイルオプション	-Kfast, openmp	-O3 -openmp -mmic -align array64byte	-O3 -openmp -ipo -xAVX -align array32byte

本研究では、各環境において表 1 に示す 1 ソケットを用いて計算を実施した。1.でも述べたように、MPI プロセス数を 1 とし、ソケット内を OpenMP によりスレッド並列化したプログラムを実行している。表 1 に示すように実際に使用したスレッド数は、FX10 : 16, MIC : 240, IvyB : 10 である。IvyB の有効スレッド数は 20 であるが、今回は 10

として実施している。

## 4. スレッド並列化, 最適化に関連する項目

### 4.1 色づけによるリオーダーリング

ハイブリッド並列プログラミングモデルでは、各ノード (ソケット) に対応した局所データを OpenMP などのマルチスレッド的な手法によって並列化に処理する。ICCG 法では不完全コレスキー分解、前進代入、後退代入のプロセスでメモリへの書き込みと参照が同時に生じ、データ依存性が発生する可能性がある。これを回避するための方法として色づけ (coloring) によるリオーダーリング (reordering) が広く使用されている [1,2]。お互いに依存性を持たない要素群を同じ色に色づけることによって、色内での並列処理が可能となる。

本研究では、並列性に優れたマルチカラー法 (Multicoloring, MC) とより安定した収束を示す Reverse Cuthill-McKee (RCM) 法を組み合わせ、RCM 法に Cyclic マルチカラー法 (Cyclic Multicoloring, CM) を適用した CM-RCM 法を使用した [1,2,4]。図 2 は CM-RCM 法による並び替え例である。ここでは、4 色に色分けされており (CM-RCM(4))、たとえば、RCM の第 1, 第 5, 第 9, 第 13 組の要素群が CM-RCM 法の第 1 色に分類される。各色には 16 の要素が含まれる。CM-RCM 法における色数は、各色内の要素が依存性を持たない程度に大きい必要がある。

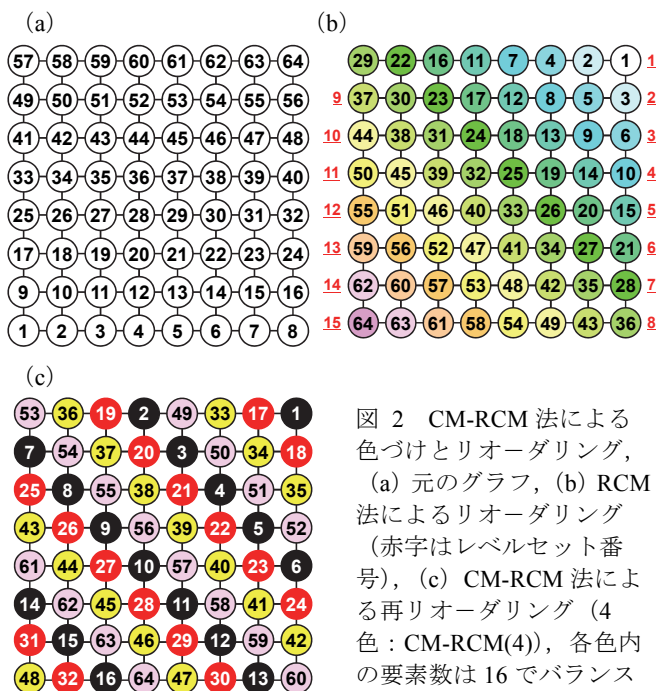


図 2 CM-RCM 法による色づけとリオーダーリング、(a) 元のグラフ、(b) RCM 法によるリオーダーリング (赤字はレベルセット番号)、(c) CM-RCM 法による再リオーダーリング (4 色 : CM-RCM(4))、各色内の要素数は 16 でバランス

### 4.2 Sequential Reordering によるデータ再配置

4.1 で示した CM-RCM 法による並び替えでは、図 3 (a) に示すように：

- 同一の色に属する要素は独立であり、並列に計算可能
- 「色」の順番に各要素を番号付けする

- 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。このような番号付けを **Coalesced Numbering** と呼ぶ。Sequential Reordering は CM-RCM による Coalesced Numbering に対して再番号付けを適用し、同じスレッドで処理するデータを連続に配置するように更に並び替えるものである（図 3 (b)）。Sequential Reordering は元々 NUMA アーキテクチャ向けの最適化手法の一つであるが [5]、本研究で扱う FX10 のような UMA アーキテクチャにも有効であることが示されており、特に色数が多い場合の効果は顕著である [1,2]。

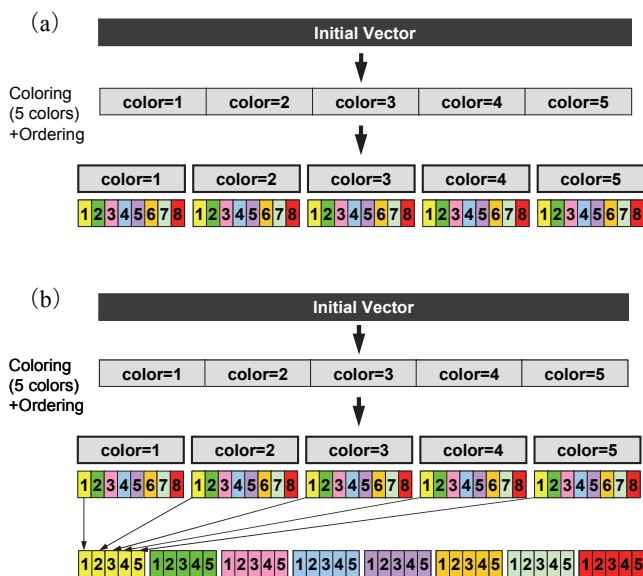


図 3 要素の番号付け (a) CM-RCM 法による番号付け (Coalesced Numbering), (b) Sequential Reordering による再番号付け (各スレッド上の要素は連続な番号付け)

### 4.3 疎行列格納形式

疎行列計算は間接参照を含むため memory-bound なプロセスである。従って疎行列演算において、演算性能と比較してメモリ転送性能の低い昨今の計算機の性能を引き出すことは困難である。係数行列の格納形式が性能に影響することは広く知られており、様々な手法が提案されている。

Compressed Row Storage (CRS) 形式は、図 4 (a) に示すように疎行列の非零成分のみを記憶する方法である。Ellpack-Itpack (ELL) 形式は各行における非零非対角成分数を最大非零非対角成分数に固定する方法であり（図 5 (b)）、実際に非零非対角成分が存在しない部分は係数=0として計算する。CRS と比較して高いメモリアクセス効率が得られることが知られているが、計算量、必要記憶容量ともに増加する。

これまで、行列格納形式に関する研究は行列ベクトル積に関するものが主であったが、著者等は IC 法、ILU 法

(Incomplete LU Factorization, 非対称行列向けの前処理手法) 等の前処理のようなデータ依存性を有するプロセスについて検討を実施している [1,2,6]。差分法に見られるような規則正しいメッシュでは、各行における非零非対角成分数がほぼ固定されているため、その性質を適用することが可能である。本研究で対象としている図 1 に示すような形状では、辞書的な初期番号付けにおいては、上三角成分（自分より番号の大きい隣接要素）、下三角成分（自分より番号の小さい隣接要素）の最大数は各要素において最大 3 であり、容易に ELL 形式を適用できる。スレッド並列化のためのリオーダーリングに RCM 法を適用した場合もこの関係は変わらない [1]。また、CM-RCM 法を適用した場合は、図 5 に示すように、総色数を NC とすると以下のようになることがわかっている [1,6]：

- 第 1 色：下三角成分数：0，上三角成分数：最大 6
- 第 2 色～第 (NC-1) 色：上下三角成分ともに最大 3
- 第 NC 色：下三角成分数：最大 6，上三角成分数：0

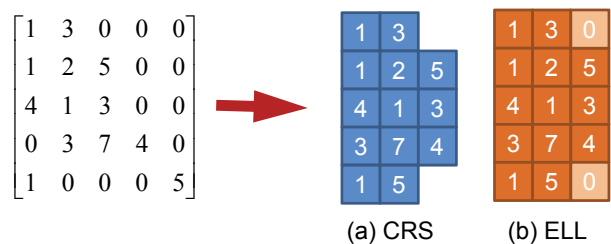


図 4 疎行列の格納形式 (a) CRS, (b) ELL

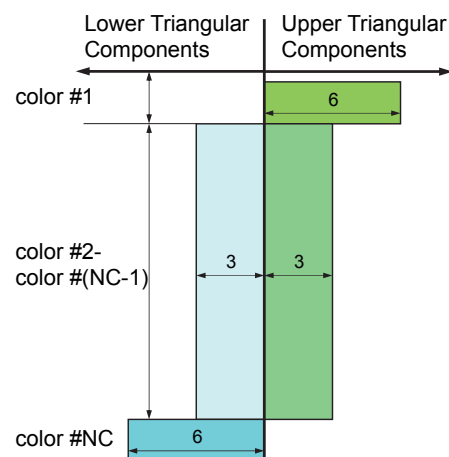


図 5 CM-RCM 法における上下三角成分数 (NC : 総色数)

著者等の先行研究 [1,2,5] では ELL 形式を適用する場合に外側ループを行方向、内側ループを列方向とする Row-wise な手法を適用してきた（図 6）。図 5 に示すようなやや不規則行列に適用する場合、無駄な計算を避けるためには、非零非対角成分の数の順番に並び替え、ループ長を変化させる手法が考えられる。図 7 に示す例では、非零

非対角成分が4以上の要素(赤)と3以下の要素(青)に分類する。図6の例に基づけば、赤い部分は「 $k=1,6$ 」、青い部分は「 $k=1,3$ 」とすることができる。

ただしこのような手法は、やや非効率的であり、図7の青い部分を計算する場合に  $A_{New}(4, i) \sim A_{New}(6, i)$  が例えキャッシュに載っていたとしても棄却されてしまう。そこで、ELL形式を拡張し、複数の配列を使用して、より効率的な計算を実施する手法として、Sliced-ELL形式 [7] が提案されている(図8)。本研究でも Sliced-ELL形式を採用している [1,2,6]。

```

!$omp parallel
do icol=1, NCOLORTot
!$omp do
do ip = 1, PEsmptTOT
do i= Index(ip-1, icol)+1, Index(ip, icol)
do k= 1, 6
Z(i)= Z(i) - AML(k, i)*Z(IAML(k, i))
enddo
Z(i)= Z(i) / DD(i)
enddo
enddo
!omp end parallel
    
```

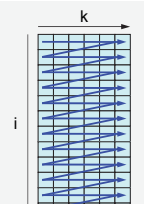


図6 ELL形式の前進代入への適用例 (Row-wise), 非零非対角成分の最大数=6. NCOLORTot: 総色数, PEsmptTOT: 総スレッド数, Index(ip, icol): 各色, スレッドに属する要素総数, AML(k, i): 非零非対角成分, IAML(k, i): 非零非対角成分(列番号), DD(i): 対角成分.

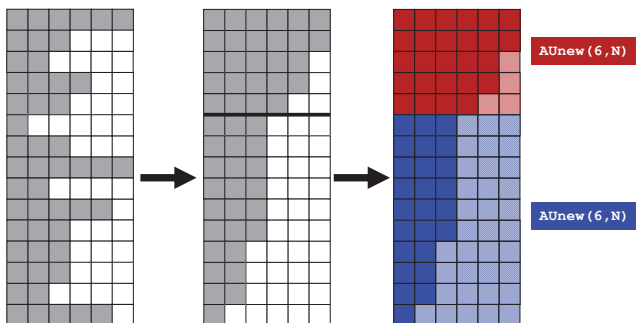


図7 ELL形式の不規則行列への適用例

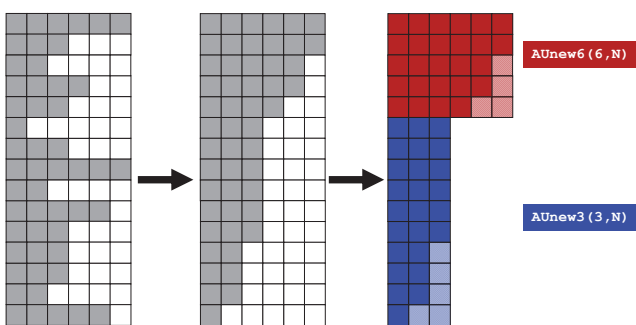


図8 Sliced-ELL形式の例 [6]

#### 4.4 Row-wise, Column-wise

図6に示した、外側ループ:行方向, 内側ループ:列方向, とする Row-wise な手法の他, 外側と内側のループを入れ替えた Column-wise な手法(図9)は, 内側ループ長を長くとることができるため, ベクトル計算機向けの手法として広く使用されて来た [8]。近年はメニコア向けの手法として再び注目されている。本研究では従来の

Row-wise 手法の他, このような Column-wise 手法についても検討するものとする。

図9に示すように, 不完全コレスキー分解, 前進・後退代入のプロセスは各色, 各スレッドに対応したブロック単位で実施されるため, Column-wise な手法では, 係数行列のアクセスが不連続となる可能性がある。本研究では, 図10に示すように, 係数行列を各色, 各スレッドに対応したブロック毎に記憶することによって, ブロック内での連続アクセスを実現した場合についても考慮した。

```

!$omp parallel
do icol=1, NCOLORTot
!$omp do
do ip = 1, PEsmptTOT
do k= 1, 6
do i= Index(ip-1, icol)+1, Index(ip, icol)
Z(i)= Z(i) + AML(i, k)*Z(IAML(i, k))
enddo
do i= Index(ip-1, icol)+1, Index(ip, icol)
Z(i)= Z(i) / DD(i)
enddo
enddo
!omp end parallel
    
```

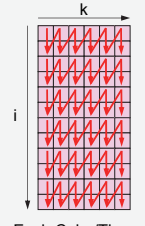


図9 ELL形式の前進代入への適用例 (Column-wise), 非零非対角成分の最大数は6としてある。NCOLORTot: 総色数, PEsmptTOT: 総スレッド数, Index(ip, icol): 各色, スレッドに属する要素総数, AML(i, k): 非零非対角成分, IAML(i, k): 非零非対角成分(列番号), DD(i): 対角成分. 各色, スレッドに対応したブロックにおいて計算が実施される(本図では各ブロックのサイズは2としてある)。

```

!$omp parallel
do icol=1, NCOLORTot
!$omp do
do ip = 1, PEsmptTOT
blkID= (ip-1)*NCOLORTot + ip
do k= 1, 6
do i= IndexB(ip-1, blkID, icol)+1, &
IndexB(ip, blkID, icol)
locID= i - IndexB(ip-1, blkID, icol)
Z(i)= Z(i) +
AMLb(locID, k, blkID)* X(IAMLb(locID, k, blkID)) &
enddo
do i= IndexB(ip-1, blkID, icol)+1, IndexB(ip, blkID, icol)
Z(i)= Z(i) / DD(i)
enddo
enddo
!omp end parallel
    
```




図10 ELL形式の前進代入への適用例 (Column-wise, ブロック化), 非零非対角成分の最大数は6としてある。NCOLORTot: 総色数, PEsmptTOT: 総スレッド数, blkID: ブロック ID, IndexB(ip, blkID, icol): 各色, スレッドに属する要素総数, locID: ブロック内要素番号, AMLb(locID, k, blkID): 非零非対角成分, IAMLb(locID, k, blkID): 非零非対角成分(列番号), DD(i): 対角成分. 各色, スレッドに対応したブロックにおいて計算が実施される(本図では各ブロックのサイズは2としてある)。

#### 4.5 Sliced-ELLにおける記憶容量削減

著者等の先行研究 [1,2,6] では, 図8に示すような Sliced-ELL形式を図5に示すような係数行列による疎行列ベクトル積に適用する際, 図11に示すように, 全要素数を  $N$  とすると全ての係数行列について  $(6, N)$ ,  $(3, N)$  のように全要素数に対応したサイズの記憶要領を確保している。しかしながら, Sliced-ELL形式で使用する配列の種類が増加すると, 必要な記憶容量が爆発的に増加する可能性があ

る。そこで、各配列について必要な容量だけを確保する場合についても検討した。図12はAU\_6, AU\_3, AL\_3, AL\_6, についてそれぞれ必要な分だけ記憶領域を確保した実装例である。記憶容量を削減できるが、その分係数行列に対する間接参照が必要となるため、計算効率が低下する可能性がある。

図11, 12 に示したのは Row-wise な場合であるが、Column-wise の場合は上下成分とも非零非対角成分数を6として、一種類の配列を使用した。

```

do icol= 1, NCOLORTot
  if (icol.eq.1) then
    do i= COLORindex(icol-1)+1, COLORindex(icol)
      do k= 1, 6
        Y(i) = Y(i) + AU_6(k, i)*X(IAU_6(k, i))
      enddo
    enddo
  else if (icol.le.NCOLORTot-1) then
    do i= COLORindex(icol-1)+1, COLORindex(icol)
      do k= 1, 3
        Y(i) = Y(i) + AU_3(k, i)*X(IAU_3(k, i)) &
          + AL_3(k, i)*X(AL_3(k, i))
      enddo
    enddo
  else
    do i= COLORindex(icol-1)+1, COLORindex(icol)
      do k= 1, 6
        Y(i) = Y(i) + AL_6(k, i)*X(AL_6(k, i))
      enddo
    enddo
  enddo
enddo
    
```

図11 Sliced-Ell形式による疎行列ベクトル積(図5)の例 (Row-wise) (従来手法)

```

do icol= 1, NCOLORTot
  if (icol.eq.1) then
    do i= COLORindex(icol-1)+1, COLORindex(icol)
      do k= 1, 6
        Y(i) = Y(i) + AU_6(k, i)*X(IAU_6(k, i))
      enddo
    enddo
  else if (icol.le.NCOLORTot-1) then
    do i= COLORindex(icol-1)+1, COLORindex(icol)
      i0= i - COLORindex(i)
      do k= 1, 3
        Y(i) = Y(i) + AU_3(k, i0)*X(IAU_3(k, i0)) &
          + AL_3(k, i0)*X(AL_3(k, i0))
      enddo
    enddo
  else
    do i= COLORindex(icol-1)+1, COLORindex(icol)
      i0= i - COLORindex(NCOLORTot-1)
      do k= 1, 6
        Y(i) = Y(i) + AL_6(k, i0)*X(AL_6(k, i0))
      enddo
    enddo
  enddo
enddo
    
```

図12 Sliced-Ell形式による疎行列ベクトル積(図5)の例 (Row-wise) (記憶容量削減型), i0: 2色目以降で使用される各配列のID

## 5. 計算例

### 5.1 実施ケースの概要

本研究では、以下の各項目に着目して実施ケースを設定した:

- Numbering (Coalesced, Sequential (図3))
- 行列格納形式 (CRS, Sliced-ELL (図4, 7, 8))
- 外側ループ (Row-wise, Column-wise (図6,9))
- Column-wiseにおけるブロック化 (図10)
- Sliced-ELLにおける記憶容量削減効果 (図12)

表2に実施ケースを示す。本研究では、図1において  $NX=NY=NZ=128$ , 総メッシュ数 2,097,152 の場合について検討を実施した。

表2 実施ケースの概要

	Numbering	行列格納形式	外側ループ	その他
AR-0	Coalesced (図3(a))	CRS	行方向 (Row-wise, 図6)	
AR-1		Sliced-ELL		記憶容量削減型(図12)
AR-2			列方向 (Column-wise, 図9)	ブロック化 (図10)
AC-1				
AC-2				
BR-0	Sequential (図3(b))	CRS	行方向 (Row-wise, 図6)	
BR-1		Sliced-ELL		ブロック化 (図10)
BC-1			列方向 (Column-wise, 図9)	
BC-2				

### 5.2 計算結果

図13に計算結果を示す。図13(a)はCM-RCMの色数と  $\epsilon=10^{-8}$  とした場合の収束までの反復回数, 図13(b)は表2におけるAR-1, BR-1 (Sliced-ELL, Row-wise) の場合のICCGソルバーの計算時間 (IC分解の時間を除く (以下同様)) である。

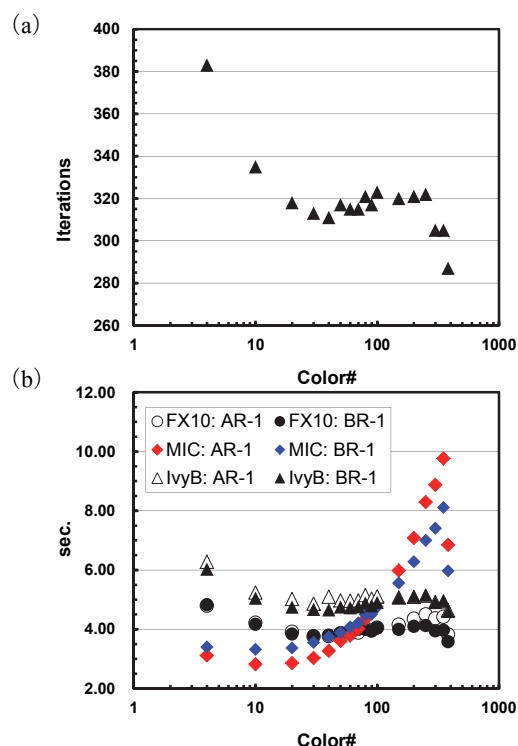


図13 ICCG法ソルバーの計算性能 (AR-1, BR-1: Sliced-ELL, Row-wise), 要素数:  $128^3 (=2,097,152)$  (a) 色数と反復回数の関係 (b) 色数と計算時間の関係

色数とともに反復回数は減少するが、同期のオーバーヘッドが増すため、特にMICでは計算時間が増大する。3つ

の計算機環境の中では CM-RCM(10)の MIC のケースが最も計算時間が短い. FX10, IvyB は CM-RCM(382),すなわち RCM の場合が最も計算時間が短い.

図 14 は CM-RCM(4)~CM-RCM(100)について, 各計算機環境における, 全ケースの ICCG 法ソルバーの計算時間を示したものである. FX10 と MIC においては CRS (AR-0, BR-0) と Sliced-ELL の差異は明らかであり, Sliced-ELL の効果は大きい. FX10 は MIC と比較すると色数による性能の影響が小さい. IvyB は色数による変動は FX10 同様少ないが, 各ケース間における差異も FX10, MIC と比較して小さい.

図 14 は FX10 (CM-RCM(30)), MIC (CM-RCM(10)), IvyB (CM-RCM(40)) における, 各ケースの計算結果 (ICCG 法ソルバーの計算時間) である. 各計算機環境における結果の分析については次節以降に詳しく述べる.

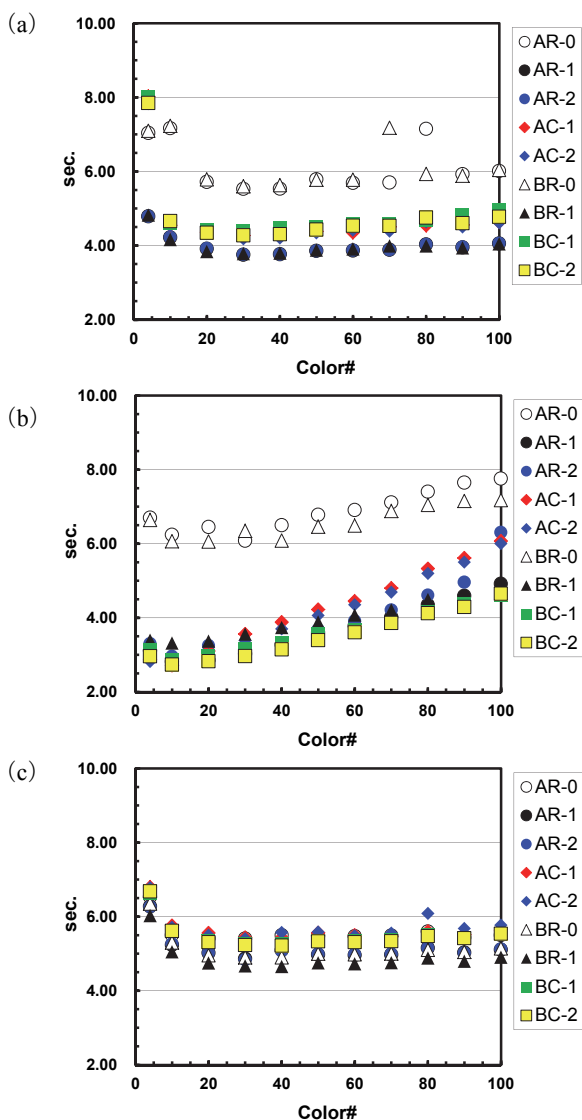


図 13 ICCG 法ソルバーの計算性能 (色数と計算時間の関係), 要素数:  $128^3 (=2,097,152)$  (a) FX10, (b) MIC, (c) IvyB

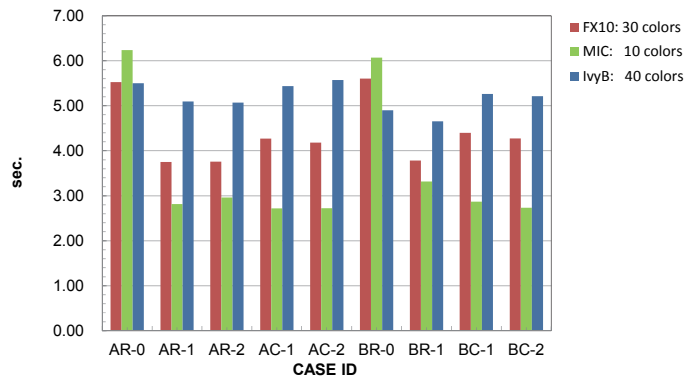


図 14 ICCG 法ソルバーの計算性能 (計算時間), 要素数:  $128^3 (=2,097,152)$ , FX10: CM-RCM(30), MIC: CM-RCM(10), IvyB: CM-RCM(40)

### 5.3 FX10

表 3 は Fujitsu PRIMEHPC FX10 専用詳細プロファイラ [3]によって算出した CM-RCM(30)における性能である.

図 15 はこれらの中で特に高い性能を示している, AR-1, AR-2, BR-1 における色数と計算性能の関係である. 図 14, 15 と表 3 を参照することによって以下のことがわかる.

- ELL の効果は大きい (AR-1, AR-2, BR-1)
- Column-wise な場合 (AC-1, AC-2, BC-1, BC-2) の性能は Row-wise と比較して低い. これは表 3 に示す総命令数, L1D ミス数とも関連している.
- AC-2, BC-2 は, AC-1, BC-1 と比較して性能が向上しており, Column-wise な場合のブロック化 (図 10) の効果は認められるものの, Row-wise と比較して性能は低い.
- AR-1 と AR-2 を比較するとほぼ性能は一致しており, 図 12 に示す Sliced-ELL 形式向けの記憶容量削減のための実装に起因する係数行列に対する間接参照による性能低下はほとんど無い.
- Coalesced (AR-1, AR-2) と Sequential (BR-1) を比較すると, 図 14, 表 3 に示す範囲では, ほぼ同じ性能を示し, AR-1, AR-2 がやや速いものの, 100 色を超えると BR-1 の性能が卓越し, RCM における最大性能においても BR-1 が上回っている. これは著者による先行研究 [1] において既に得られている知見である.

表 4 は, Sequential Numbering を適用した場合, BR-0(CRS), BR-1 (Sliced-ELL) の性能を, 図 7 に示す ELL 形式を適用した場合と比較したものである (CM-RCM(30)の場合). ELL 形式 (図 7) と Sliced-ELL の計算量は等しく, 総命令数も同じであるが, ELL 形式はキャッシュミスが多いため, GFLOPS 値としては CRS (BR-0) よりもむしろ低くなっていることがわかる.

表3 Fujitsu PRIMEHPC FX10 専用詳細プロファイラによる ICCG 法ソルバーの性能評価 (FX10, CM-RCM(30)) (要素数:  $128^3 (=2,097,152)$ )

	GFLOPS 計算時間: sec.	メモリスルー プット (GB/sec)	総命令数	L1D ミス数	L2 ミス数
AR-0	5.64 5.53	46.3	$1.93 \times 10^{11}$	$2.30 \times 10^9$	$1.71 \times 10^9$
AR-1	6.84 3.75	62.8	$6.41 \times 10^{10}$	$1.88 \times 10^9$	$1.56 \times 10^9$
AR-2	6.83 3.76	62.7	$6.41 \times 10^{10}$	$1.89 \times 10^9$	$1.56 \times 10^9$
AC-1	6.13 4.27	60.0	$6.71 \times 10^{10}$	$2.67 \times 10^9$	$1.64 \times 10^9$
AC-2	6.27 4.18	62.0	$6.74 \times 10^{10}$	$2.56 \times 10^9$	$1.66 \times 10^9$
BR-0	5.56 5.61	43.8	$1.93 \times 10^{11}$	$2.31 \times 10^9$	$1.64 \times 10^9$
BR-1	6.78 3.79	62.4	$6.41 \times 10^{10}$	$1.88 \times 10^9$	$1.56 \times 10^9$
BC-1	5.96 4.40	59.1	$6.71 \times 10^{10}$	$2.69 \times 10^9$	$1.66 \times 10^9$
BC-2	6.12 4.28	60.0	$6.74 \times 10^{10}$	$2.56 \times 10^9$	$1.66 \times 10^9$

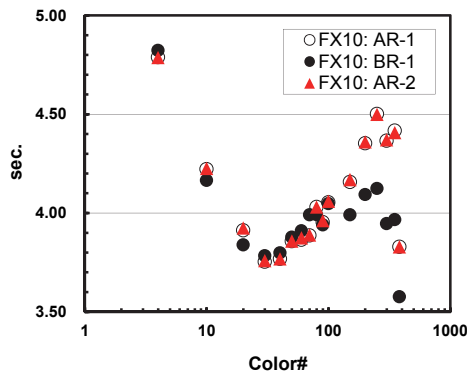


図 15 ICCG 法ソルバーの計算性能 (色数と計算時間の関係) (FX10), 要素数:  $128^3 (=2,097,152)$

表4 Fujitsu PRIMEHPC FX10 専用詳細プロファイラによる ICCG 法ソルバーの性能評価, ELL 形式 (図 7) との比較 (FX10, CM-RCM(30)) (要素数:  $128^3 (=2,097,152)$ )

	GFLOPS 計算時間: sec.	メモリスルー プット (GB/sec)	総命令数	L1D ミス数	L2 ミス数
CRS (BR-0)	5.56 5.61	43.8	$1.93 \times 10^{11}$	$2.31 \times 10^9$	$1.64 \times 10^9$
ELL (図 7)	5.02 5.12	64.5	$6.41 \times 10^{10}$	$2.66 \times 10^9$	$2.29 \times 10^9$
Sliced ELL (BR-1)	6.78 3.79	62.4	$6.41 \times 10^{10}$	$1.88 \times 10^9$	$1.56 \times 10^9$

#### 5.4 MIC

図 16 は, 図 13 (b) から CRS (AR-0, BR-0) の結果を除いたものである。図 14, 16 から MIC については以下の知見が得られる:

- ELL の効果は大きい (AR-1, AR-2, BR-1)
- Column-wise な場合 (AC-1, AC-2, BC-1, BC-2) の性能は Row-wise と比較して全般的に高い。これは FX10, IvyB とは異なる傾向である。Column-wise はベクトル化が効きやすいため, 特に MIC で高い性能が得られていると考えられる。
- AC-2, BC-2 は, AC-1, BC-1 と比較して性能が向上しており, Column-wise な場合のブロック化の効果は顕著ではないものの認められる。BC-2 では効果がやや大きい。
- AR-1 と AR-2 を比較するとほぼ性能は一致しているが, 係数行列に対する間接参照を含む AR-2 は色数が 80 色以上になると性能が急激に低下する。
- Coalesced (AC-2) と Sequential (BC-2) は最適値 (CM-RCM(10)) では, AC-2 の性能がわずかに高いものの, 他の色数では概して BC-2 の性能が良い。全体的に AR-1, BC-1, BC-2 が安定している。

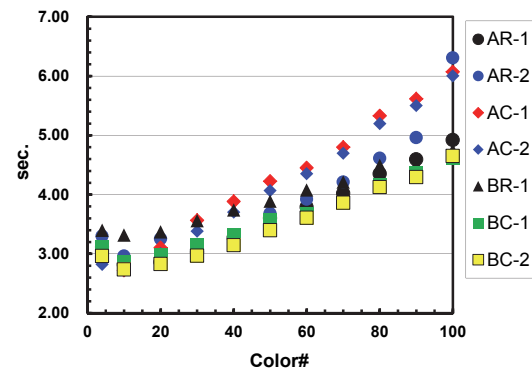


図 16 ICCG 法ソルバーの計算性能 (MIC) (色数と計算時間の関係), 要素数:  $128^3 (=2,097,152)$

#### 5.5 IvyB

図 17 は図 13 (c) の縮尺を変えたものである。BR-1 (Sequential, Row-wise) の性能が最も良く, AR-1, AR-2 (Coalesced, Row-wise) がこれに続く。Column-wise な場合の性能はこれらと比較して低い。

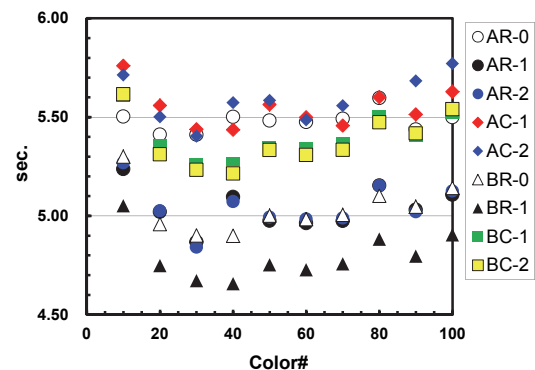


図 17 ICCG 法ソルバーの計算性能 (色数と計算時間の関係: 100 色まで) (IvyB), 要素数:  $128^3 (=2,097,152)$

図 18 は更に CM-RCM(382) (=RCM) まで表示したものであるが、FX10 と同様、BR-1 に CM-RCM(382) (=RCM) を適用したケースが最も性能が良いことがわかる。

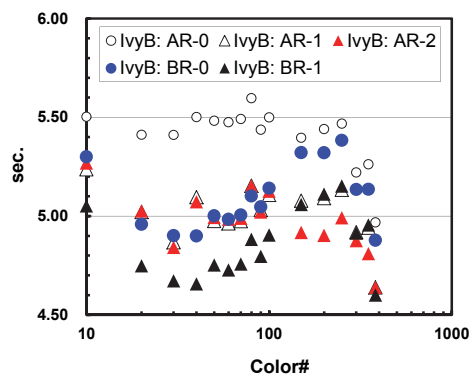


図 18 ICCG 法ソルバーの計算性能 (色数と計算時間の関係) (IvyB), 要素数:  $128^3 (=2,097,152)$

## 6. まとめ

本研究では、ELL 形式による格納手法をスレッド並列化された ICCG 法ソルバー向けに拡張し、Intel Xeon Phi, Fujitsu FX10, Intel Ivy-Bridge 上で性能評価を実施した。三次元有限体積法によるポアソン方程式ソルバーから得られる対称正定な行列を係数行列とする連立一次方程式を対象として、様々な比較検討を実施した。データ依存性を回避する並列化抽出のための色付け手法としては CM-RCM 法を使用した。

各計算機環境で ELL 形式による係数格納の効果は大きい。Fujitsu FX10, Intel Ivy-Bridge が Sequential, Row-wise な実装において、色数を最大限多くした RCM 法を適用する場合に最適な性能が得られたのに対して、Intel Xeon Phi では色数を少なめにした CM-RCM 法で、Sequential, Column-wise な実装がより高い性能を示した。

本研究では、一種類の問題しか扱わなかったため、今後、様々な問題サイズでの検討が必要である。幅広い計算機環境に対応していくためには、Sequential/Coalesced, Row/Column-wise の各手法を開発、最適化していく必要がある。

特に、有限要素法等から得られるより複雑で不規則な行列に対応していくためには、本研究でも検討した記憶容量削減型の Sliced-ELL (AR-2 に相当) が重要である。Sequential, Column-wise への適用も含めて更なる検討を実施していきたい。

**謝辞** 本研究実施にあたって貴重な助言を頂いた成瀬彰氏 (NVIDIA), 片桐孝洋准教授, 塙敏博特任准教授, 大島聡史助教 (東京大学情報基盤センター) に、謹んで感謝の意を表する。

## 参考文献

- 1) 中島研吾, 前処理付きマルチスレッド並列疎行列ソルバー, 情報処理学会研究報告 (HPC-139-6) (2013)
- 2) 大島聡史, 松本正晴, 片桐孝洋, 塙敏博, 中島研吾, 様々な計算機環境における OpenMP/OpenACC を用いた ICCG 法の性能評価, 情報処理学会研究報告 (HPC-145), (2014)
- 3) 東京大学情報基盤センター (スーパーコンピューティング部門), <http://www.cc.u-tokyo.ac.jp/>
- 4) Washio, T., Maruyama, K., Osoda, T., Shimizu, F., and Doi, S., Efficient implementations of block sparse matrix operations on shared memory vector machines. Proceedings of The 4th International Conference on Supercomputing in Nuclear Applications (SNA2000) (2000)
- 5) 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)
- 6) Nakajima, K., Optimization of Serial and Parallel Communications for Parallel Geometric Multigrid Method, Proceedings of IEEE ICPADS 2014 (in press) (2014)
- 7) Monakov, A., A. Likhomotov, and A. Avetisyan, Automatically tuning sparse matrix-vector multiplication for GPU architectures, Lecture Notes in Computer Science 5952 (2010) 112-125
- 8) Nakajima, K., Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator, ACM/IEEE Proceedings of SC2003, (2003)