

ICCG 法ソルバーの Intel Xeon Phi 向け最適化

中島研吾^{†1 †2} 大島聡史^{†1 †2} 埜 敏博^{†1} 星野哲也^{†1} 伊田明弘^{†1 †2}

SELL-C- σ 法は疎行列演算の性能を高める行列格納手法として注目されているが、これまでは専ら疎行列ベクトル積に適用されてきた。科学技術計算において広く使用されている ICCG 法は前進後退代入、不完全コレスキー分解等のデータ依存性を有するプロセスを含むため、多色順序付け等によって並列性を抽出する必要がある。本研究は世界でも初めて、ICCG 法に SELL-C- σ 法を適用した事例である。Intel Xeon Phi (Knights Corner, Knights Landing) 上での性能評価を実施し、特に Knights Landing 上では従来手法と比較して高い性能改善を達成することができた。

Optimization of ICCG Solver for Intel Xeon Phi

Kengo Nakajima^{†1 †2} Satoshi Ohshima^{†1 †2} Toshihiro Hanawa^{†1}
 Tetsuya Hoshino^{†1} Akihiro Ida^{†1 †2}

SELL-C- σ storage format is widely known method for efficient computation of sparse matrices. It has been mainly applied to SpMV operations. ICCG is used for solving linear equations with sparse matrices in a wide range of applications of science and engineering. Because ICCG includes operations with data-dependency, such as forward/backward substitutions, and incomplete Cholesky factorization, extraction of parallelism by reordering is needed. The present work is the first example, where SELL-C- σ storage format is applied to ICCG. Performance of the developed solver has been evaluated on Intel Xeon Phi (Knights Corner, Knights Landing), and performance of the ICCG with SELL-C- σ on Knights Landing was better than existing methods.

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) によってメニョコクスタ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) 形式とその拡張手法に着目する。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 (Knights Corner, Knights Landing), Intel Broadwell-EP 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 [1,2] :

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

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

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。

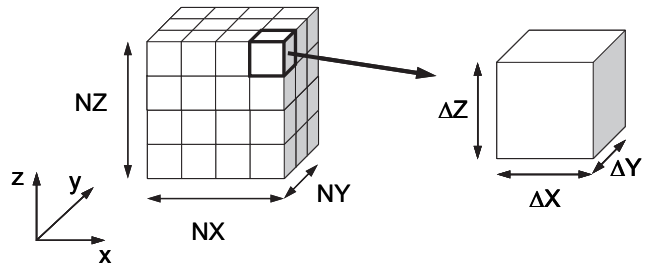


図 1 三次元ポアソン方程式ソルバーの解析対象差分格子の各メッシュは直方体 (辺長さは ΔX , ΔY , ΔZ), X, Y, Z 各方向のメッシュ数は NX, NY, NZ

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 $\{b\}$

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

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[4]は疎 (sparse) な行列となる。

係数行列[4]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]。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 [1,2]。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

3.1 概要

本研究では以下の 4 種類の計算機環境を使用した、Knights Landing としてはコア数、周波数等の異なる 2 種類の CPU (KNL-0, KNL-1) を使用している：

- **KNC** : Intel Xeon Phi 5110P (Knights Corner)
- **KNL-0** : Intel Xeon Phi 7210 (Knights Landing)
- **KNL-1** : Intel Xeon Phi 7250 (Knights Landing), 最先端共同 HPC 基盤施設 (JCAHPC) [3] の運用する Oakforest-PACS の 1 ノード
- **BDW** : Intel Broadwell-EP, 東京大学情報基盤センター [4] の運用する Reedbush-U システムの 1 ソケット

表 1 各計算環境 (1 ソケット) の概要

略 称	KNC	KNL-0	KNL-1	BDW
名 称	Intel Xeon Phi 5110P (Knights Corner)	Intel Xeon Phi 7210 (Knights Landing)	Intel Xeon Phi 7250 (Knights Landing)	Intel Xeon E5-2695 v4 (Broadwell-EP)
動作周波数 (GHz)	1.053	1.30	1.40	2.10
コア数 (最大有効スレッド数)	60 (240)	64 (256)	68 (272)	18 (18)
理論演算性能 (GFLOPS)	1,010.9	2,662.4	3,046.4	604.8
主記憶容量 (GB)	8	MCDRAM: 16 DDR4: 96	MCDRAM: 16 DDR4: 96	128
メモリバンド幅性能 (GB/sec., Stream Triad)	159	MCDRAM: 454 DDR4: 72.5	MCDRAM: 490 DDR4: 84.5	65.5

注) KNL-0,1 におけるメモリバンド幅性能は、メモリモード Flat, サブ NUMA クラスタリングモード Quadrant における性能である

プログラムは Fortran90 で記述しており、Intel Compiler (Ver.17) / Intel Parallel Studio XE 2017, OpenMP 4.0 を使用した。表 1 に計算機環境の概要を示す。

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。

3.2 Knights Landing の特徴

Knights Landing (Intel Xeon Phi 72xx) のハードウェア構成は、従来の Xeon Phi シリーズ (Knights Corner) と大きく異なる。主な違いとしては以下が挙げられる：

- 最大コア数が 72 まで増加している
- 2 コアで 1 タイルを構成し、タイルごとに L2 キャッシュを搭載している
- コアごとの処理がインオーダー実行からアウトオブオーダー実行に変更された
- コアをリング状に繋ぐ構成から、タイル間をメッシュ状に繋ぐ構成に変更された。以下のいずれかのモードを選択可能である (ハードウェア起動時に BIOS にて指定する) :
 - ✓ **SNC** : メッシュを分割して、複数ソケット CPU のような NUMA 構成として扱う
 - ✓ **Quadrant** : 全体を 1 ソケット CPU のように扱う (デフォルト)
- 通常の DDR4 メモリの他、CPU 上に高速な三次元積層メモリ MCDRAM を搭載し、メモリモードとして以下のいずれかを選択可能である (これらもハードウェア起動時に BIOS にて指定する) :
 - ✓ **Flat** : 外部に接続した DDR4 メモリと MCDRAM のメモリ空間を個別に扱う。メモリの使い分けの指示は、プログラム中で専用のメモリ確保関数、プログラム実行時の numactl 指定によって行う。
 - ✓ **Cache** : MCDRAM を DDR4 に対するキャッシュのように扱う
 - ✓ **Hybrid** : Flat モードと Cache モードのハイブリッドのいずれかを選択可能

本稿における Knights Landing の性能は、KNL-0 では Flat モードと Quadrant モードの組み合わせによる結果を用いる。ただし 5.5 においては KNL-1 を使い、Flat・Quadrant の組み合わせと Cache・Quadrant の組み合わせによる性能を比較する。

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することによって, 色内での並列処理が可能となる.

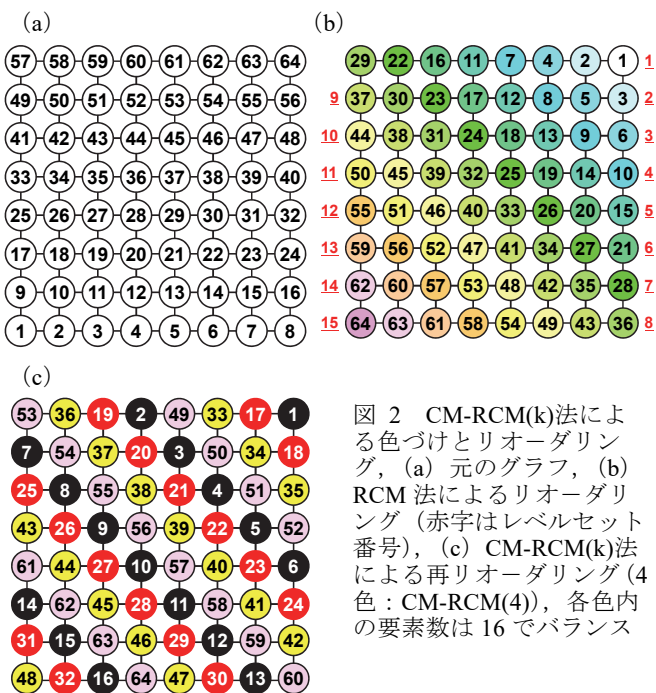


図 2 CM-RCM(k)法による色づけとリオーダーリング, (a) 元のグラフ, (b) RCM 法によるリオーダーリング (赤字はレベルセット番号), (c) CM-RCM(k)法による再リオーダーリング (4色: CM-RCM(4)), 各色内の要素数は 16 でバランス

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(k)法を使用した [1,2,5]. 図 2 は CM-RCM(k)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(k)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(k)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.

4.2 Sequential Reordering によるデータ再配置

4.1 で示した CM-RCM(k)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(k)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 [6], 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].

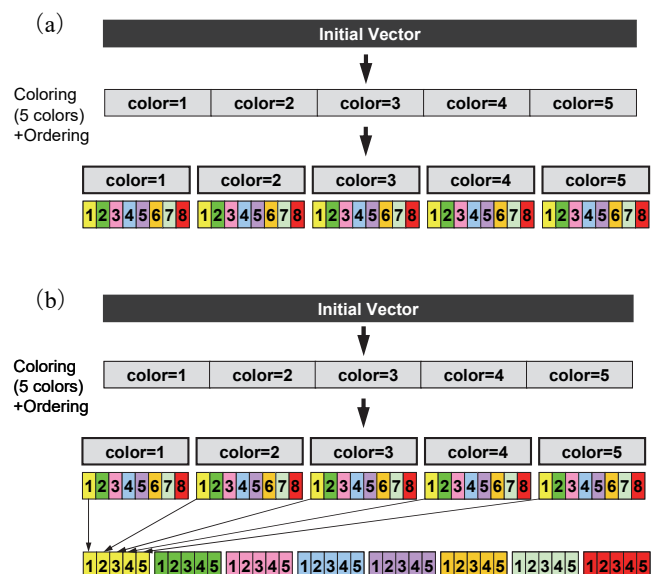


図 3 要素の番号付け (a) CM-RCM(k)法による番号付け (Coalesced Numbering), (b) Sequential Reordering による再番号付け (各スレッド上の要素は連続な番号付け)

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 [1,2,7]. 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(k)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 [2,7]:

- 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

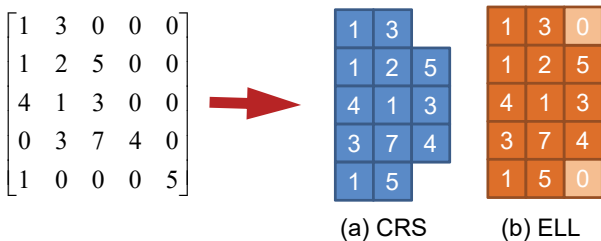


図 4 疎行列の格納形式 (a) CRS, (b) ELL

著者等の先行研究 [1,2,6] 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, やや非効率であり, 図 7 の青い部分を計算する場合に $A_{\text{new}}(4, i) \sim A_{\text{new}}(6, i)$ 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 7).

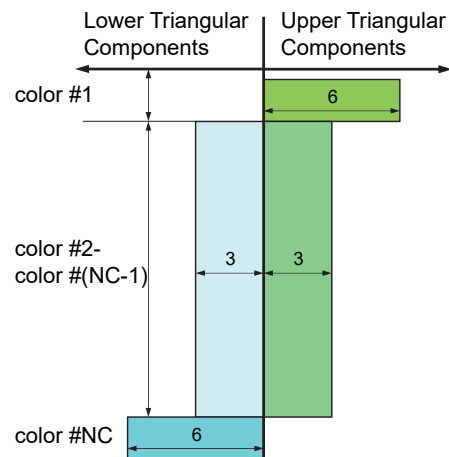


図 5 CM-RCM(k)法における上下三角成分数 (NC: 総色数)

```

!$omp parallel
do icol= 1, NCOLORTot
!$omp do
do ip = 1, PEsmptOT
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
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, PEsmptOT: 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.

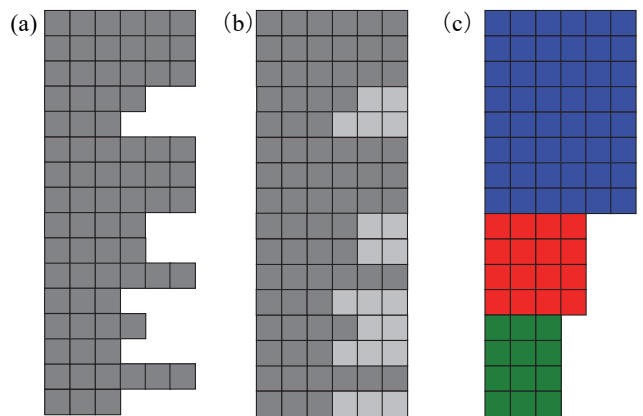


図 7 ELL 形式の不規則行列への適用例 (a) CRS, (b) ELL, (c) Sliced-ELL

4.4 Row-wise, Column-wise

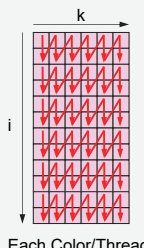
図 6 に示した, 外側ループ: 行方向, 内側ループ: 列方向, とする Row-wise な手法の他, 外側と内側のループを入れ替えた Column-wise な手法 (図 8) は, 内側ループ長を長くとることができるため, ベクトル計算機向けの手法として広く使用されて来た [9]. 近年はメニコア向けの手法として再び注目されている. 本研究では従来の Row-wise 手法の他, このような Column-wise 手法についても検討するものとする.

図 8 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9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する。

Column-wise な手法は OpenMP 4.0 以降サポートされている「!\$omp simd」を最内側ループに適用することにより、ベクトル化が効率良く適用され、高い計算性能を得られることが期待される。

```
!$omp parallel
do icol = 1, NCOLortot
!$omp do
do ip = 1, PEsmptOT
do k = 1, 6
!$omp simd
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
enddo
!omp_end_parallel
```



Each Color/Thread

図8 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, PEsmptOT: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, PEsmptOT
blkID = (ip-1)*NCOLORtot + ip
do k = 1, 6
!$omp simd
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9 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, PEsmptOT: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 SELL-C- σ

SELL-C- σ [10] は ELL 及び Sliced ELL (SELL) を SIMD プロセッサ向けに拡張した疎行列格納方式である(図10)。C (chunk size) は計算を実行する単位であり、 σ (sorting scope) は疎行列の非零非対角成分の分布によって決定されるパラメータである。図11はサイズ=2の chunk が4つで1つのグループを構成している場合で、このような場合を SELL-2-8 と呼ぶ(8=2×4)。図10の場合は非零非対角成

分数が6,6,4,3の4つの chunk があるが、各 chunk において非零非対角成分数が変わらない場合は SELL-C-1 と呼ぶ。

Intel Xeon Phi のようなアーキテクチャでは、C を SIMD 幅(倍精度実数の場合8 (=512bit/64))に設定するとベクトル化の効率が高まる。本研究では、4.4で述べた Column-wise な手法、「!\$omp simd」と組み合わせて適用する。図11は前進代入部(icol=2~NCOLORtot-1)に SELL-8-1 を適用した事例である。本研究では、padding を実施して、各色・各スレッドで処理する要素数が8で割り切れるようにしてある。

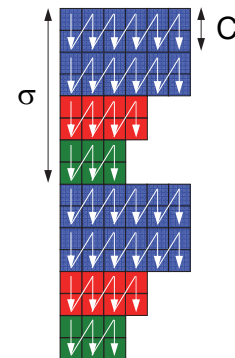


図10 SELL-C- σ 形式(10), C=2 and $\sigma=8$ の場合

```
!$omp parallel private(ic,ip,ip0,iq0,iq1,iq2,ib,ib0,is,i,k)
...
do ic = 2, NCOLortot-1
!$omp do
do ip = 1, PEsmptOT
iq1 = Index(ip-1,icol)
iq2 = Index(ip,icol)
ip0 = (ic-1)*PEsmptOT + ip
iq0 = (iq1-1)*8
do ib = iq1, iq2
ib0 = ib-iq1
do k = 1, 3
!$omp simd
do is = 1, 8
i = iq0 + ib0 + is
Z(i) = Z(i) - AMLss(is,k,ib0,ip0)*Z(IAMLss(is,k,ib0,ip0))
enddo
enddo
!$omp simd
do is = 1, 8
i = iq0 + ib0 + is
Z(i) = Z(i)/DD(i)
enddo
enddo
enddo
!omp_end_parallel
...
```

図11 SELL-8-1形式の前進代入への適用例(Column-wise, Sequential), 非零非対角成分の最大数は3としてある。NCOLORtot:総色数, PEsmptOT:総スレッド数, Index(ip,icol):各色, スレッドに属する要素総数, ip0:ブロックID(各色・各スレッド), blkID:ブロック内チャックID, AMLss(8,k,blkID,ip0):非零非対角成分, IAMLss(8,k,blkID):非零非対角成分(列番号), DD(i):対角成分。

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, SELL-C- σ (図4,7,10))

- 外側ループ (Row-wise, Column-wise (図 6,8))
- Column-wise におけるブロック化 (図 9)
- SELL-C- σ (図 11)

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. SELL-C- σ としては SELL-8-1 の場合のみ実施した.

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		列方向 (Column-wise, 図 8)	
AC-1					ブロック化 (図 9)
AC-2				SELL-8-1	
AC-3		SELL-C- σ			
BR-0	Sequential (図 3 (b))	CRS	行方向 (Row-wise, 図 6)		
BR-1		Sliced-ELL		列方向 (Column-wise, 図 8)	
BC-1					ブロック化 (図 9)
BC-2				SELL-8-1	
BC-3			SELL-C- σ		

使用したスレッド数は下記である :

- KNC : 240
- KNL-0 : 62, 124
- KNL-1 : 64, 128, 192, 124
- BDW : 18

である. KNC はコア当りのスレッド数を 4 とした, スレッド数=240 の場合が最も性能が高いが, KNL-0, KNL-1 はコア当りスレッド数=1,2 の方が一般的に性能が良い.

KNL-0, KNL-1 においてスレッド数による性能差はほとんど無いが, 最速値を採用している.

5.2 Knights Landing での実行について

5.2.1 KNL-0

KNL-0 においては Flat モードと Quadrant モードの組み合わせを用い, 基本的に MCDRAM のみを用いた場合の性能を評価している. 使用したスレッド数は 62, 124, 186, 248 スレッドである. KNL-0 は 64 コアを搭載し最大 256 スレッドでの実行が可能プロセッサであるが, スレッド ID 0 のコアにのみタイマー割り込みを行わせるような tickless 設定がなされているため, 当該スレッドに対応する 1 タイル上では計算を行わないような指定をしている. 例えば 124 スレッド実行の際には, 環境変数 OMP_NUM_THREADS と KMP_AFFINITY にそれぞれ :

- OMP_NUM_THREADS=124
- KMP_AFFINITY=granularity=fine,proclist=[2-63,66-127], explicit

という指定をしている. この proclist 指定の場合, 各コアに対するスレッドの割り当て順序は 62 のコアに 1 スレッドずつを割り当て終えてからそれぞれのコアに第 2 スレッドを割り当てるという順序になり, KMP_AFFINITY に compact や balanced を指定した場合よりも scatter を指定した場合に近い.

5.2.2 KNL-1

KNL-1 についても同様に, 68 コア・最大 272 スレッドでの利用が可能であるが, tickless 設定を考慮した 64, 128, 192, 256 スレッドで使用している.

Flat モードでプログラムを実行する際には :

- numactl --membind=1 ./a.out

という形式でプログラムを実行することで MCDRAM のみを選択的に使用している. Cache モードでプログラムを実行する場合には実行時の特別な指定は不要である.

コンパイル時の主なオプションは -align array64byte -O3 -xMIC-AVX512 -qopenmp である.

5.3 予備的計算結果 (KNC, KNL-0, BDW)

図 12 は CM-RCM(k)法の色数を変化させた場合の, ICCG 法の収束までの反復回数, 図 13 は ICCG 法による線形ソルバーの計算時間である. 図 13 は KNC, KNL-0, BDW における最適ケース (AC-3, BC-3 を除く) の結果が掲載されている. 5.2.1 で述べたように, KNL-0 では Flat/Quadrant モードの組み合わせで, 高速な MCDRAM のみを用いた場合の性能を評価しているが, 参考のため DDR4 を使用したケースも実施している.

KNC では AC-2 (Coalesced, Sliced ELL, Column-wise, ブロック化) が最も性能が高いが, 他の場合は BR-1 (Coalesced, Sliced ELL, Row-wise) が最も性能が高かった. 図 12 に示すように, 色数が増加すると反復回数は減少するが [1,2], KNC, KNL-0 ではスレッド数が多いため, 色数が増加すると同期のオーバーヘッドの影響が顕著となる. したがって, 図 13 に示すように, 色数が増加した場合でも計算時間が一般的に増加する. 特に, KNC ではこの増加が顕著である. 今回の問題では KNC, KNL-0 ともに 10 色程度が最適色数となっている. この最適色数は問題設定, ハードウェアの特性, 使用スレッド数によって変動する. 一方, BDW はスレッド数が少ないため, 色数増加による計算時間低下は他と比較して少ない.

KNL-0 (MCDRAM) は KNL-0 (DDR4) と比較して 3 倍強速い。

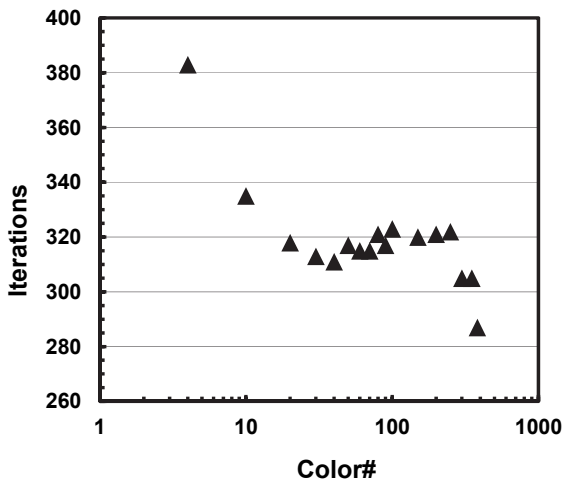


図 12 ICCG 法ソルバーの計算性能, 要素数: 128^3 ($=2,097,152$), 色数と反復回数 (b) 色数と計算時間の関係

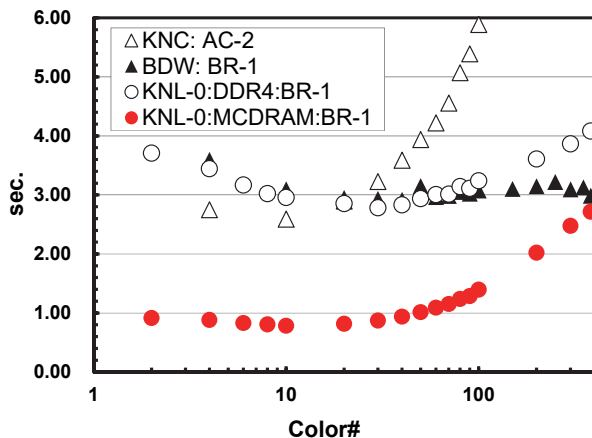


図 13 ICCG 法ソルバーの計算性能, 要素数: 128^3 ($=2,097,152$), 色数と計算時間の関係

5.4 SELL-C- σ の効果

続いて, 2 色~10 色を対象として KNC, KNL-0 において SELL-C- σ (SELL-8-1) を適用し, 性能を評価した。まず, KNC では BR-1 (Sequential, Sliced ELL, Row-wise) と比較して AC-2 (Coalesced, Sliced ELL, Column-wise, ブロック化) の性能が高いが, KNL-0 ではほとんど差はなく, BR-1 の方がやや速い。

KNC では, SELL-C- σ による効果はなく, AC-2 と比較してむしろ性能は低下している。KNL-0 でも効果は少ないが, BR-1 と比較して若干性能が向上している。

図 15 は, 各色における SELL-C- σ による性能改善率を示している。具体的には, AC-3 (Coalesced, SELL-C- σ) と AC-2 (KNC), BR-1 (KNL-0) の ICCG 法計算時間の比率である。KNL-0 では, 2 色~10 色で 5% 以上の性能向上が見られるが, KNC では 2 色以外では概して性能が低下している。SELL-C- σ では padding により計算量が若干増加するとともに, ループ構造が元のアルゴリズムと比較して複雑

なることも起因していると考えられる。

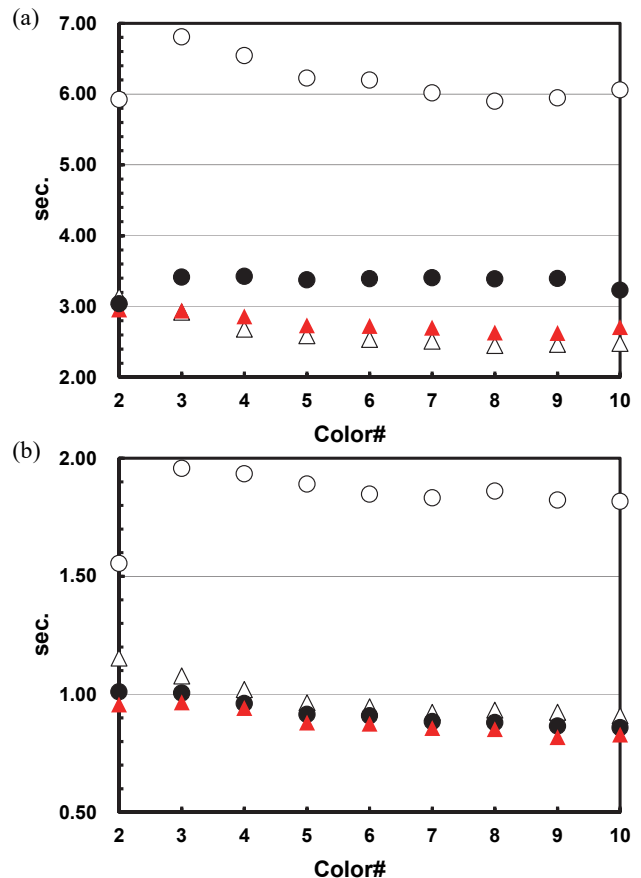


図 14 ICCG 法ソルバーの計算性能, 要素数: 128^3 ($=2,097,152$), 色数と計算時間の関係 (a) KNC, (b) KNL-0, ○: AR-0, ●: BR-1, △: AC-2, ▲: AC-3 (Coalesced, SELL-C- σ)

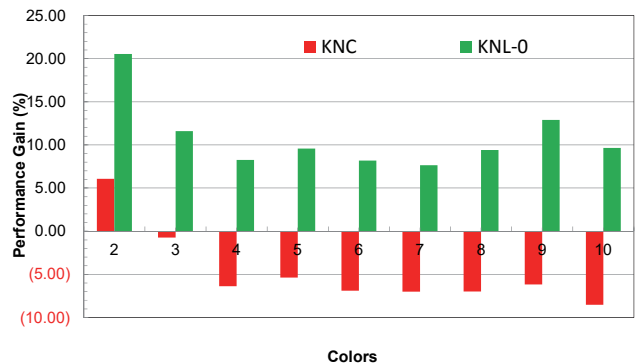


図 15 ICCG 法ソルバーの計算性能, 要素数: 128^3 ($=2,097,152$), 各色における AC-3 (Coalesced, SELL-C- σ) の AC-2 (KNC), BR-1 (KNL-0) に対する性能改善率

また非零非対角成分数が少ないことも SELL-C- σ による性能改善が少ない要因の一つと考えられる。図 5 からわかるように, 2 色の場合, 第 1 色では, 非零非対角成分は全て上三角成分 (行当り最大 6 個), 第 2 色の場合は全て下三角成分 (行当り最大 6 個) となっており, 各グループで処理する非零非対角成分の数が多い。この場合, 図 15 にも示すように KNC, KNL-0 とともに性能改善率は高く, KNL-0 では 20% を超えている。色数が大きくなると色番号が 2~

NC-1に含まれる要素の数が多くなる。このような要素は上下三角成分の数が行当たりそれぞれ最大3個と2色の場合と比較して少ないため、性能改善率も少ないものと考えられる。

5.5 MCDRAMの利用モードによる性能への影響(KNL-1)

KNLに搭載された高速メモリMCDRAMの利用モードがプログラムの実行にどのような影響を及ぼすかを確認するため、FlatモードとCacheモードそれぞれで同一のプログラムを実行して性能を確認した。本節ではKNL-1を用いて実験を行った。

一般的に対象プログラムがMCDRAMに収まりきるサイズの場合はFlatモードを用いればよく、Cacheモードを用いるメリットは特にない。本稿の対象としている問題も16GBのMCDRAMに十分収まる規模である。我々はむしろCacheモードを用いることによる性能低下の有無や大小に着目している。これは、動的に問題サイズが決まるような問題を想定した場合に、問題サイズがMCDRAMの容量を超える場合でもCacheモードを用いれば計算が行える一方、問題サイズが小さい場合にもCacheモードを用いるとFlatモードよりも有意に性能が低下するのであれば、問題サイズにより実行環境の選択・切替をするべきであり、そのような利用方法を行うべきかを見積もるためという意味がある。

図16は、KNL-0において良い性能の得られたBR-1 (Sequential, Sliced ELL, Row-wise) (10色)について、KNL-1上で64, 128, 192, 256スレッドそれぞれのFlatモードとCacheモードの性能を測定した結果(実行時間および実行時間比)を示している。いずれも10回実行時の最速値である。測定の結果、64スレッド実行ではCacheモード、それ以外のスレッド数ではFlatモードが高速であったが、その実行時間の割合は最大でも3.5%と小さい。この結果からは、MCDRAMに収まるメモリ容量の問題を扱う場合でも、Flatモードに設定せずCacheモードで実行しても大き

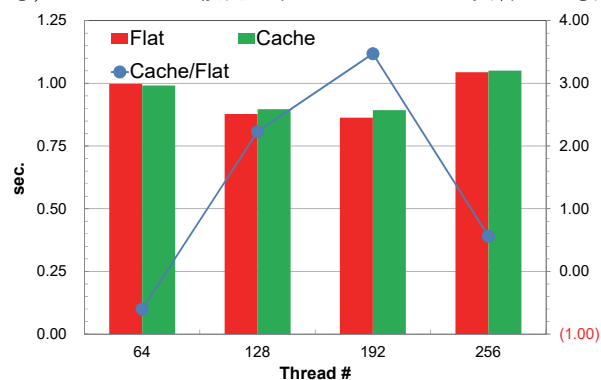


図16 FlatモードとCacheモードの実行時間および実行時間比、要素数: $128^3 (=2,097,152)$, BR-1 (Sequential, Sliced ELL, Row-wise) (10色), KNL-1における実行

な性能ペナルティは生じないと考えられる。

6. まとめ

SELL-C- σ 法は疎行列演算の性能を高める行列格納手法として注目されているが、これまでは専ら疎行列ベクトル積に適用されてきた。本研究ではデータ依存性を含むICCG法にSELL-C- σ 法を適用した事例である。Intel Xeon Phi (Knights Corner, Knights Landing) 上での性能評価を実施し、特にKnights Landing上では従来手法と比較して高い性能改善を達成することができた。しかしながら、Sliced ELLと比較した性能改善率は数%であり、更なる最適化とともに、様々なアプリケーションへの適用による検証が必要である。

更にIntel Xeon Phi (Knights Landing) の挙動を確認するため、MCDRAMに収まるメモリ容量の問題をFlatモードとCacheモードで実行し性能を比較した結果、その時間差は3.5%以下と非常に小さかった。

参考文献

- 1) 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)
- 2) 中島研吾, 拡張型Sliced-ELL行列格納手法に基づくメニコア向け疎行列ソルバー, 情報処理学会研究報告 (HPC-147) (2014)
- 3) 最先端共同 HPC 基盤施設, <http://jcahpc.jp/>
- 4) 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/>
- 5) 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)
- 6) 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)
- 7) Nakajima, K., Optimization of Serial and Parallel Communications for Parallel Geometric Multigrid Method, Proceedings of IEEE ICPADS 2014 (in press) (2014)
- 8) Monakov, A., A. Lokhmotov, and A. Avetisyan, Automatically tuning sparse matrix-vector multiplication for GPU architectures, Lecture Notes in Computer Science 5952 (2010) 112-125
- 9) 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)
- 10) Kreuzer, M., Hager, G., Wellein, G., Fehske, H. and Bishop, A.R.: A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units. SIAM Journal on Scientific Computing 36-5 (2014) C40