Vol.2020-HPC-174 No.5
2020/5/13

低精度演算とアプリケーション性能

中島研吾[†1 †2] 坂本龍一[†1] 星野哲也[†1] 有間英志[†1] 塙敏博[†1] 近藤正章[†1 †2]

近年，科学技術計算において，低精度演算を積極的に活用することにより，計算時間を短縮する試みが活発に行われている．また，低精度演算による計算の精度を保証するための実用的手法についても研究が進められている．本研究では，アプリケーションの実装方法，問題規模と低精度演算による性能改善の関係に注目し，様々なハードウェア環境下での検討を実施した．

1. はじめに

著者等は，有限体積法によるポアソン方程式ソルバー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）によってメニィコアクラスタで効率よく解くための研究を実施し，GPU を含む様々な環境での評価を実施している〔1,2,3〕．近年は，低精度演算に着目し，倍精度・単精度演算の比較，消費電力，消費エネルギーの測定を実施し，条件の比較的良い問題では，単精度演算により倍精度の場合と比較して，50～60%程度の計算時間，消費エネルギーで正しい計算を実施することができた〔4,5〕．

これまで実施してきた研究は，低精度演算の有効性を示すための予備的研究であるが，本研究では，問題サイズ，行列格納手法，対象アーキテクチャなどの様々な影響を考慮した検討を実施した．

本稿では以下，本研究の背景，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. 高性能・変動精度・高信頼性数値解析手法とその応用

エクサスケールシステムにおける，高性能数値アルゴリズム実現のためには，通信最適化が必須であり，様々な提案がなされているが，通信最適化にはノード間通信の最適化（Parallel Optimization）の他，メモリアクセスの最適化（Serial Optimization）も含まれており，本稿でもメモリアクセス最適化のために様々な疎行列格納法に関する検討を実施している．

もう一つの技術的課題は消費電力，エネルギー（以下「消費電力」）である．ハードウェア的技術革新と共に，省電力・省エネルギー（以下「省電力」）のためのアルゴリズムの開発，実用化により，実計算時の消費電力の抑制が期待される．Approximate Computing〔6〕は，低精度演算の積極的活用により計算時間短縮，消費電力削減を図る試みである．混合精度演算はその一種であり，既に多くの研究事例があるが，Approximate Computing では，半精度から四倍精度まで演算精度を動的に変動させる変動精度（Transprecision）の研究が進められている．数値計算による近似解（数値解）は様々な計算誤差を含み，計算結果の信頼性の観点から，数値解の正しさを数学的に保証する必要があり，低精度・変動精度使用時，悪条件問題には重要である．昨今は，スパコンによる大規模計算向け精度保証の研究も実施されているが，実問題で現れる大規模疎行列・H 行列（階層化行列，密行列を低ランク近似等により階層化する手法）系への応用例はほとんどない．著者等はこのような背景に基づき，学際大規模情報基盤共同利用・共同研究拠点 2018 年度共同研究課題「高性能・変動精度・高信頼性数値解析手法とその応用（課題番号: jh20037-NAH）」〔7〕に取り組んでいる．当該研究は，最先端のスパコン向けに開発された高性能数値アルゴリズムに対して，半精度から倍精度，倍々精度までの広範囲をカバーする変動精度演算を適用し，精度保証，そのための自動チューニング手法を開発する新しい試みであり，開発された手法を様々なアプリケーションに適用することで，低精度を中心とした変動精度演算の科学技術シミュレーションへの有効性の検証を目的としている．開発したアルゴリズム，アプリケーションの消費電力の直接測定によって，各計算の特性と低精度演算の有効性を消費電力の観点から明らかにすることを目指している〔4,5〕．

3. 対象とするアプリケーション

3.1 概要

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〕：

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

$$\phi = 0 @ z = z_{\max} \tag{2}$$

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

†1 東京大学情報基盤センター
Information Technology Center, The University of Tokyo
†2 理化学研究所計算科学研究センター
RIKEN，Center for Computational Science (R-CCS)

造格子型のデータとして考慮する．

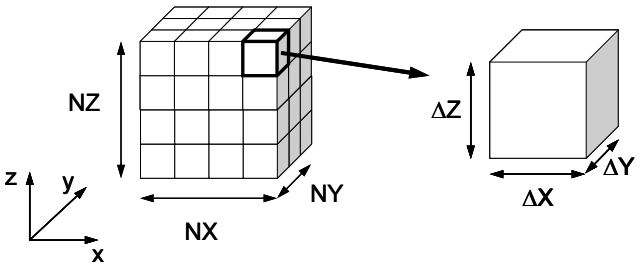

図 1　三次元ポアソン方程式ソルバーの解析対象
差分格子の各メッシュは直方体（辺長さはΔX，ΔY，ΔZ），
X，Y，Z 各方向のメッシュ数は NX，NY，NZ

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：

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

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〕．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があるため，リオーダリングが必要である〔1,2〕．

3.2 色づけによるリオーダリング

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リング，（a）元のグラフ，（b）RCM 法によるリオーダリング（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〔1,2,8,9〕．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．本研究では k=10 としている．

3.3 Sequential Reordering によるデータ再配置

3.2 で示した CM-RCM(k)法による並べ替えでは，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〔1,2〕，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〕.

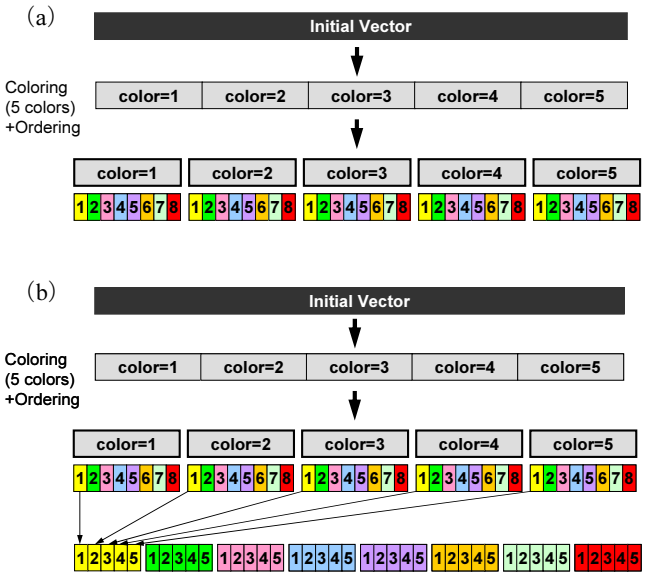

図 3 要素の番号付け（a）CM-RCM(k)法による番号付け（Coalesced Numbering），（b）Sequential Reordering による再番号付け（各スレッド上の要素は連続な番号付け）

3.4 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〔1,2,9〕．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〔1,2〕．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〔1,2〕：

- 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

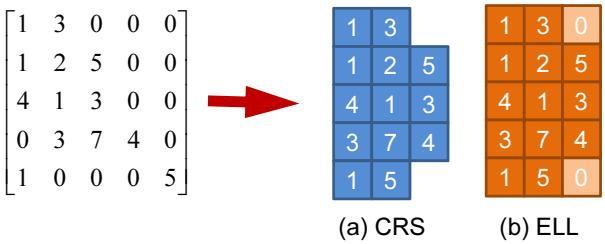

図 4 疎行列の格納形式（a）CRS，（b）ELL

著者等の先行研究〔1,2〕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 `AUnew(4,i)`～`AUnew(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〔10〕が提案されている（図 7）.

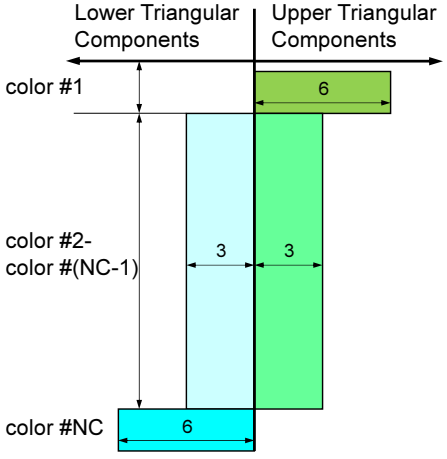

図 5 CM-RCM(k)法における上下三角成分数（NC：総色数）

```
!$omp parallel
    do icol= 1, NCOLORtot
!$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
```

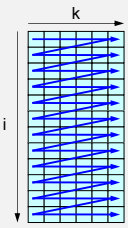

図 6　ELL 形式の前進代入への適用例（Row-wise），非零非対角成分の最大数=6．NCOLORtot：総色数，PEsmpTOT：総スレッド数，Index(ip,icol)：各色，スレッドに属する要素総数，AML(k,i)：非零非対角成分，IAML(k,i)：非零非対角成分（列番号），DD(i)：対角成分．

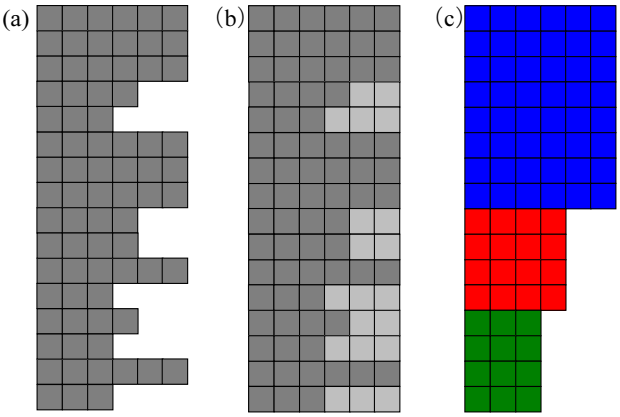

図 7　ELL 形式の不規則行列への適用例 (a) CRS，(b) ELL，(c) Sliced-ELL

3.5 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〕．近年はメニィ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．

```
!$omp parallel
    do icol= 1, NCOLORtot
!$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
      enddo
      do i= Index(ip-1,icol)+1, Index(ip,icol)
        Z(i)= Z(i) / DD(i)
      enddo
    enddo
    enddo
!omp end parallel
```

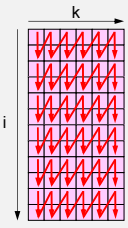

図 8　ELL 形式の前進代入への適用例（Column-wise），非零非対角成分の最大数は 6 としてある．NCOLORtot：総色数，PEsmpTOT：総スレッド数，Index(ip,icol)：各色，スレッドに属する要素総数，AML(i,k)：非零非対角成分，IAML(i,k)：非零非対角成分（列番号），DD(i)：対角成分．各色，スレッドに対応したブロックにおいて計算が実施される（本図では各ブロックのサイズは 2 としてある）．

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
      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
      enddo
      do i= IndexB(ip-1,blkID,icol)+1, IndexB(ip  ,blkID,icol)
        Z(i)= Z(i) / DD(i)
      enddo
    enddo
    enddo
!omp end parallel
```

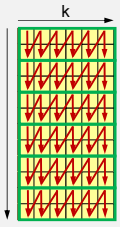

図 9　ELL 形式の前進代入への適用例（Column-wise，ブロック化），非零非対角成分の最大数は 6 としてある．NCOLORtot：総色数，PEsmpTOT：総スレッド数，blkID：ブロック ID，IndexB(ip,blkID,icol)：各色，スレッドに属する要素総数，locID：ブロック内要素番号，AMLb(locID,k,blkID)：非零非対角成分，IAMLb(locID,k,blkID)：非零非対角成分（列番号），DD(i)：対角成分．各色，スレッドに対応したブロックにおいて計算が実施される（本図では各ブロックのサイズは 2 としてある）．

3.6 SELL-C-σ

SELL-C-σ〔11〕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．

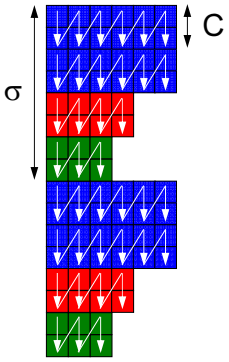

図 10　SELL-C-σ形式〔10〕，C=2 and σ=8 の場合〔11〕

```
!$omp parallel private(ic,ip,ip0,iq0,iq1,iq2,ib,ib0,is,i,k)
...
      do ic= 2, NCOLORtot-1
!$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
      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），chkID：ブロック内チャンク ID，AMLss(8,k,chkID,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．

4. 計算機環境

本研究では以下の 5 種類の計算機環境を使用した：

マルチコア・メニィコア CPU

① Reedbush-U（東京大学情報基盤センター）〔12〕
② OFP-Mini（東京大学情報基盤センター）〔12〕
③ Oakbridge-CX（東京大学情報基盤センター）〔12〕

GPU

④ Reedbush-L（東京大学情報基盤センター）〔12〕
⑤ P9-V100 クラスタ（東京大学情報基盤センター）

本研究では，①～③については各 1 ノード，④・⑤については GPU1 基を使用して実施した．表 1 は①～③の各 1 ソケット，表 2 は④・⑤の各 GPU の概要である．

②の OFP-Mini は，最先端共同 HPC 基盤施設（JCAHPC）の運用する Oakforest-PACS と同じ CPU を使用した 8 ノードのクラスタである．また，⑤は，IBM Power 9 と NVIDIA Tesla V100（V100）から構成されるクラスタである．

表 1　本研究で使用した計算機（CPU）各ソケット概要

システム名	Reedbush-U	OFP-mini	Oakbridge-CX
略称	RBU	OFP	OBCX
CPU 名称	Intel Xeon E5-2695 v4 (Broadwell-EP)	Intel Xeon Phi 7250 (Knights Landing, KNL)	Intel Xeon Platinum 8280 (Cascade Lake, CLX)
ソケット当たりコア数	18	68	28
ノード当たりソケット数	2	1	2
理論演算性能（GFLOPS）	604.8	3,046.4	2,419.2
主記憶容量（GB）	128	MCDRAM: 16 DDR4: 96	96
メモリ性能（GB/sec）STREAM Triad [14])	65.5	MCDRAM: 490 DDR4: 84.5	101.0
TDP (W)	120	215	205

表 2　本研究で使用した GPU 概要

システム名	Reedbush-L	P9 Cluster
略称	P100	V100
GPU 名称	NVIDIA Tesla P100	NVIDIA Tesla V100
理論演算性能（GFLOPS）	5,300	7,500
主記憶容量 (GB)	16	32
メモリ性能（GB/sec）STREAM Triad [14])	557	855
TDP (W)	250	300

元のプログラムは Fortarn+OpenMP によって開発されているが，GPU 用にはこれに OpenACC を適用したバージョン〔3〕を使用している．

電力評価には RBU，OBCX，P100，V100 については Intel が提供する CPU の電力制御機能の RAPL〔15〕を使用した．また OFP については，同じく Intel の提供する Intel pcm-power.x を使用している．

5. 計算例

5.1 実施ケースの概要

本研究では，以下の 3 種類の行列格納形式について検討した：

- Compressed Row Storage（CRS）（図 4）
- Column-Wise な Sliced ELL（ELL）（図 7，9）〔10〕
- SELL-C-σ（C=8, σ=1）（SCS）（図 10，11）〔11〕

また，RBU，OFP，OBCX については SCS を更に最適化してスレッド並列化によるオーバーヘッドを取り除いた実装〔2,3〕を適用した．本稿ではこれを「OPT」と呼ぶ．

RBU，OBCX については 1 ノード（2 ソケット）を使用した．RBU では合計 36 コア（36 スレッド），OBCX では 48 コア（48 スレッド）を使用した．また OFP では，先行研究〔1,2〕における最適な設定である 64 コア（128 スレッド）を使用した．P100，V100 は GPU1 基を使用した．

OFP は表 1 に示すように，通常の DDR4 メモリに加えて，CPU パッケージに直接収められている高速の MCDRAM を使用することが可能である．Cache モードでは，MCDRAM を DDR4 のキャッシュとして使用ができる他，Flat モードでは，DDR4 と MCDRAM を使いわけることが可能である．本研究では，以下の 3 ケースについて検討を実施した：

- a: Cache モード
- b: Flat モード，DDR4 のみ使用
- c: Flat モード，MCDRAM のみ使用

問題規模としては：

- Tiny（t）： 32,768 DOF（=32^3）
- Small（s）： 262,144 DOF（=64^3）
- Medium（m）： 2,097,152 DOF（=128^3）
- Large（l）： 16,777,216 DOF（=256^3）

の 4 種類を実施した．Large の場合でも，P100，OFP（MCDRAM）の各メモリに充分収まっている．

本研究では，全計算を倍精度で実施した場合（Double）と全て単精度実施した場合（Single）の 2 種類の計算を実施した．単精度のみの場合は，倍精度の場合と比較して 20%程度反復回数が増加している〔4,5〕．

5.2 計算結果

図 12 は単精度を使用して実施した ICCG 法部分の計算時間について，各計算機環境，各問題サイズにおける，「倍精度・CRS」の計算時間によって無次元化したものである．値が 1.00 より大きい場合は，「倍精度・CRS」より ICCG 法部分の計算時間が短いことを示している．図 12（a）は Tiny，Small，図 12（b）は Medium，Large の場合である．OFP については，Flat モード，MCDRAM のみを使用した，ケース c（OFP/c）の場合の結果である．全般的に問題規模が小さい場合には，単精度適用による速度向上の効果は少なく，問題規模が大きくなるほど顕著となる傾向がある．また OFP については，CRS では単精度による速度向上が得られず，むしろ遅くなっているが，ELL，SCS，OPT では 2 倍以上の速度向上が得られている．特に，「単精度・OPT」

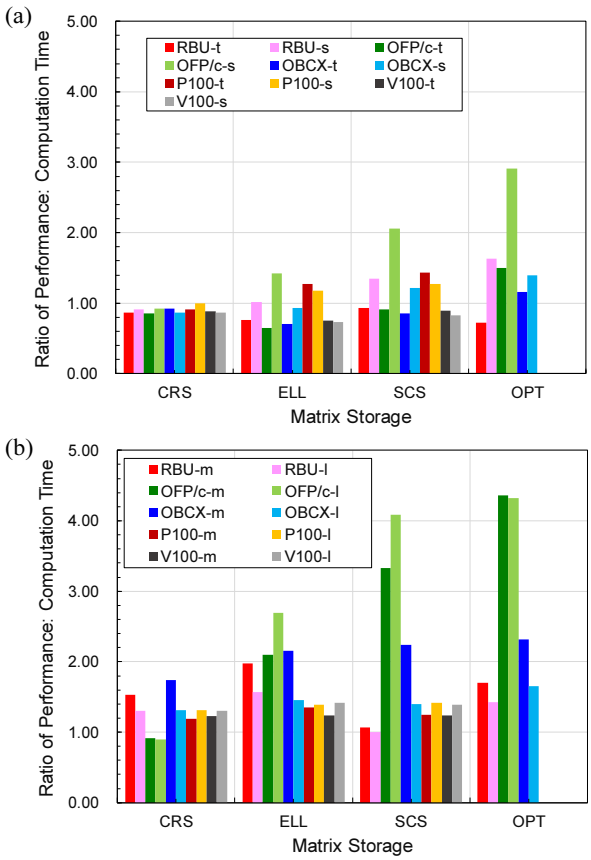

図 12 ICCG 法ソルバーの計算性能，「倍精度・CRS」の計算時間によって無次元化，(a) Tiny，Small，(b) Medium，1.00 より大きい場合は「倍精度・CRS」よりも ICCG 法の計算時間が短い

の場合には問題規模が Medium，Large では，AVX512 によるベクトル化の効果が顕著であり，反復回数が 20%増加しているにもかかわらず，「倍精度・CRS」の場合の 4 倍以上の速度向上が得られている．RBU では ELL⇒SCS で一旦性能が低下しているが，OPT によるスレッド並列化のオーバーヘッド除去により再び向上している．OBCX，P100，V100 では，ELL，SCS，OPT（OBCX のみ）の性能はほぼ同じ傾向である．OFP では全般的に問題規模増加ともに速度向の効果が顕著であるが，OBCX の場合は，問題規模増加とともにむしろ速度向上は低下している．図 13 は各ケースにおいて単純に単精度と倍精度の計算時間を比較しているが図 12 と同様の傾向が見られる．

図 14 は，ICCG 法部分の消費電力（W）について，各計算機環境，各問題サイズにおける，「倍精度・CRS」の計算時間によって無次元化したものである．図 14（a）は Tiny，Small，図 14（b）は Medium，Large の場合である．P100，V100 は計算時間が短く，Tiny，Small で有意な値を得ることができなかった．無次元化された消費電力の値は概ね 1.00 程度であるが，単精度と倍精度を直接比較すると，単精度の場合の方が，演算密度が増加するため若干大きくなる傾向がある〔4,5〕．本研究では，特に GPU（P100，V100）

の場合にその傾向が見られた．全般的に問題規模が増加すると図 15 に示すように消費電力（W）は増加するが，OFP/c（単精度）の場合，SCS と OPT では Medium での増加が低く抑えられている（図 14（b），図 15（a））．

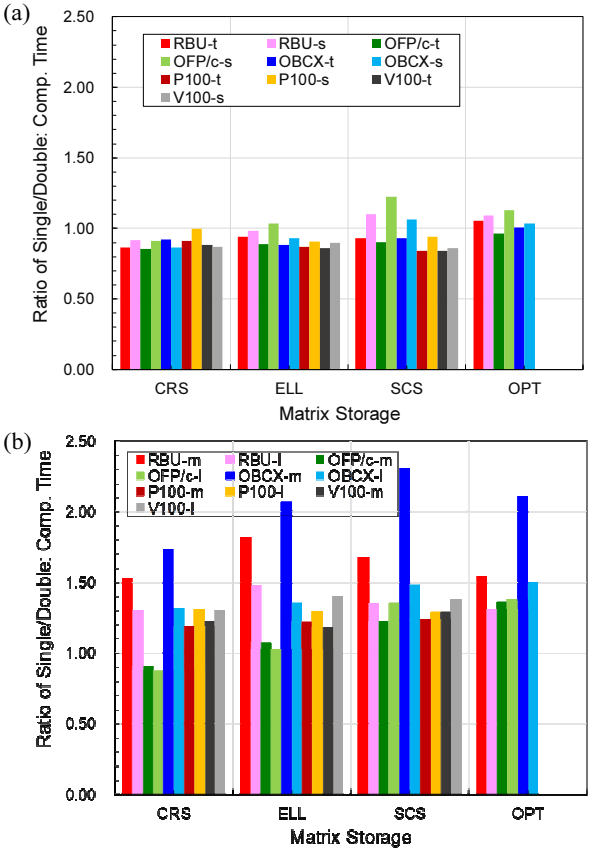

図 13　ICCG 法ソルバーの計算性能，「倍精度」計算時間によって無次元化，（a）Tiny，Small，（b）Medium，1.00 より大きい場合は「単精度」の計算時間が短い

図 15 は RBU，OFP/c，OBCX，P100，V100 における問題サイズ「Medium」の場合の ICCG 法の（a）消費電力（W），（b）消費エネルギー（J）と計算時間である．消費エネルギーと計算時間はほぼ正比例していることがわかる．計算時間としては，OFP/c，OBCX，P100，V100 はほぼ拮抗しているが，OBCX, V100 がやや速い．OBCX は消費電力（W），消費エネルギ（J）が OFP/c，P100，V100 の 3 倍以上である．OFP/c，P100，V100 は消費エネルギ（J）はほぼ等しいが，P100，V100 についてはホストの CPU の消費電力，消費エネルギーは考慮されていない．

図 16，図 17 は OFP において，

- a:　Cache モード
- b:　Flat モード，DDR4 のみ使用
- c:　Flat モード，MCDRAM のみ使用

を Medium（図 16），Large（図 17）の問題サイズを適用し，

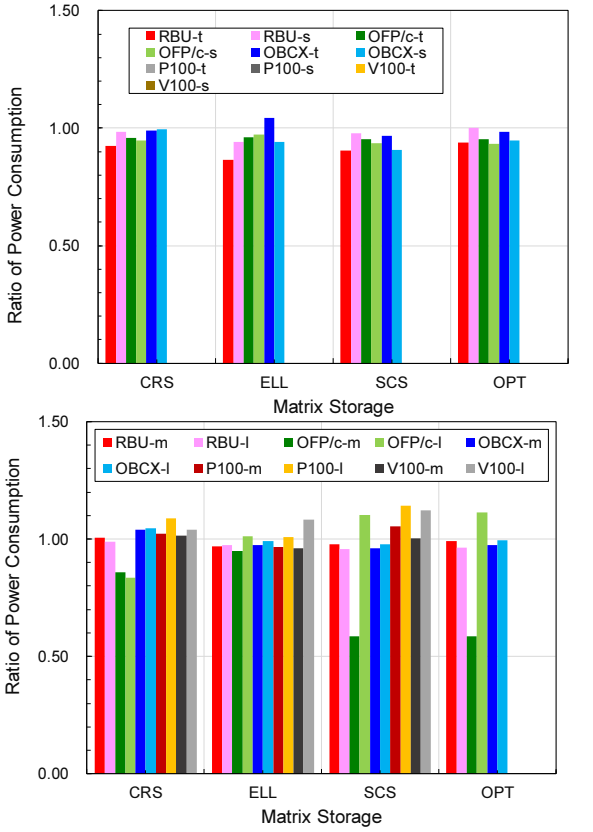

図 13　ICCG 法ソルバーの消費電力，「倍精度・CRS」の消費電力（W）によって無次元化，（a）Tiny，Small，（b）Medium，1.00 より大きい場合は「倍精度・CRS」よりも消費電力（W）が大きい

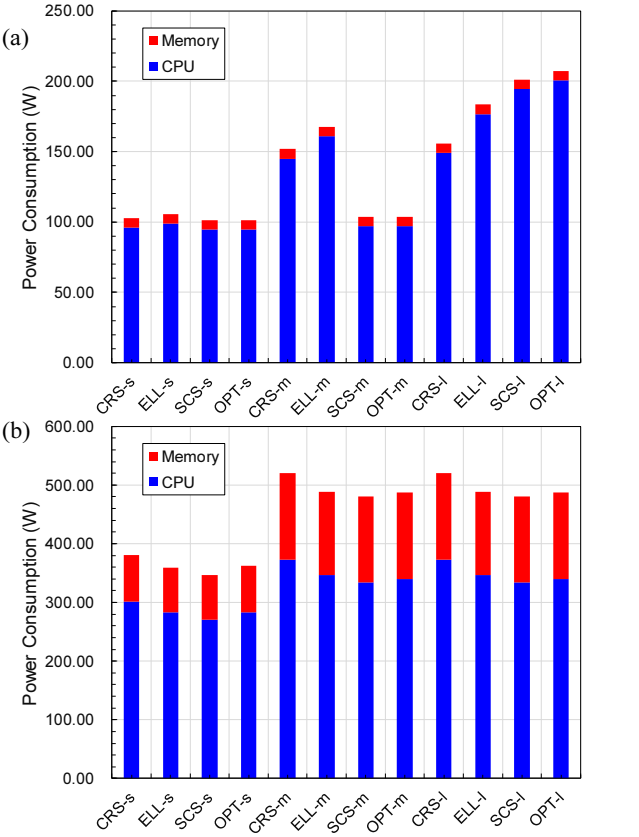

図 14　ICCG 法ソルバーにおける消費電力（W）（単精度）（a）OFP/c，（b）OBCX

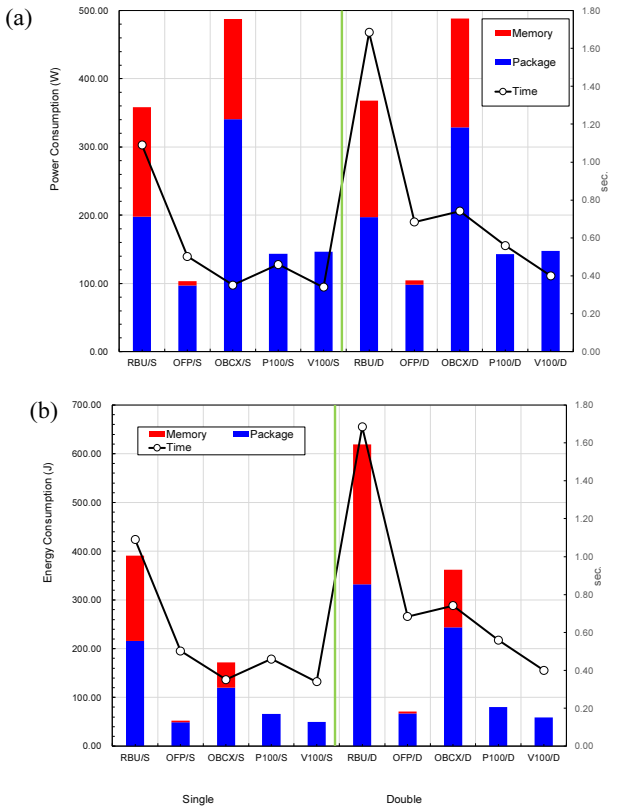

図 15 問題サイズ「Medium」の場合の ICCG 法の計算時間と（a）消費電力（W），（b）消費エネルギー（J）

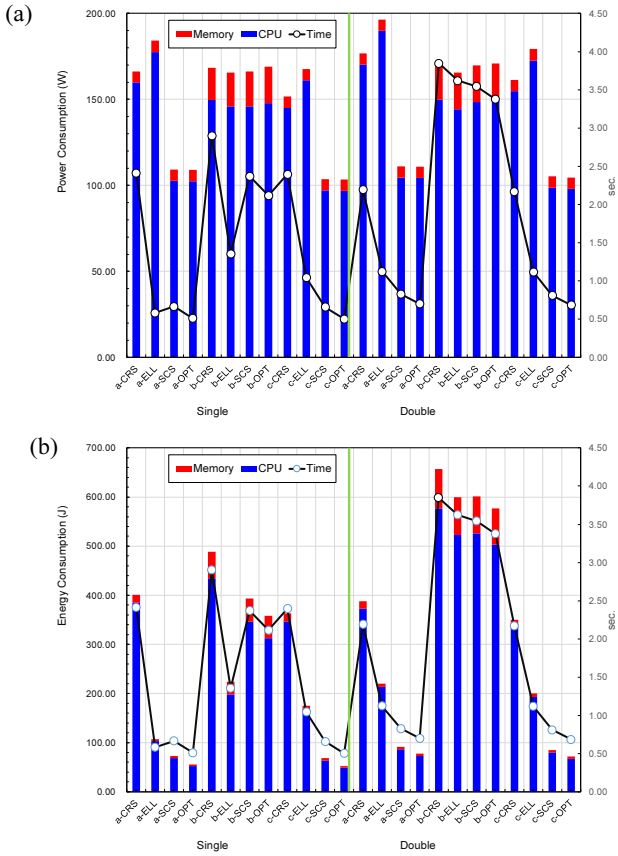

図 16 OFP：問題サイズ「Medium」の場合の ICCG 法の計算時間と（a）消費電力（W），（b）消費エネルギー（J）

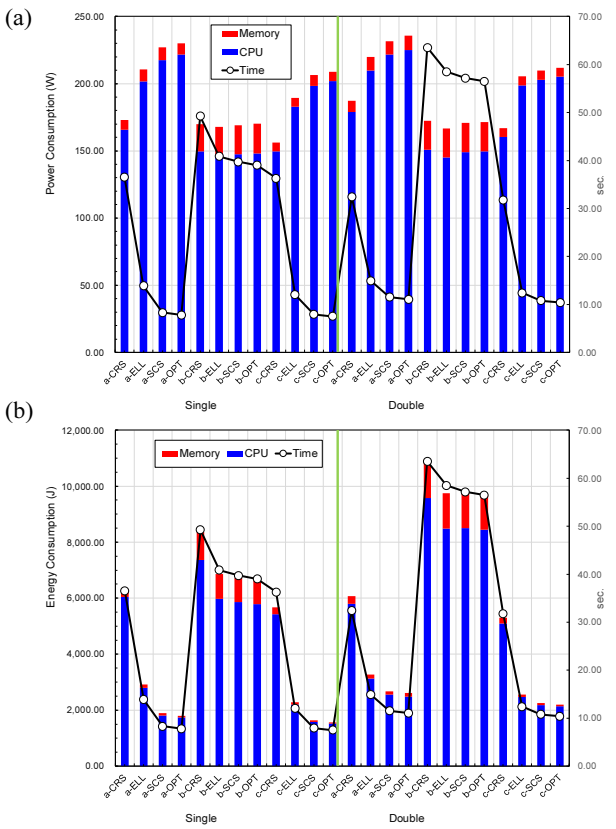

図 17 OFP：問題サイズ「Large」の場合の ICCG 法の計算時間と（a）消費電力（W），（b）消費エネルギー（J）

ICCG 法の計算時間，消費電力（W），消費エネルギー（J）を比較したものである．Cache モード（OFP/a）と Flat/MCDRAM（OFP/c）はほぼ同じ挙動である．これらのケースでは基本的に CPU 部分に収納されている MCDRAM が主として使用されるため，メモリ部分の電力消費はほとんどない．

問題サイズが大きくなると消費電力（W）はむしろ OFP/b が他と比べて小さくなるが，表 1 に示すように MCDRAM と DDR のメモリバンド幅の日は 6:1 程度であるため，本研究で扱っている疎行列ソルバーのような memory-bound なアプリケーションでは，その影響は顕著である．OFP/c-OPT と OFP/b-OPT の計算性能の比は倍精度の場合 5 倍程度である．

6. まとめ

近年，科学技術計算において，低精度演算を積極的に活用することにより，計算時間を短縮する試みが活発に行われている．また，低精度演算による計算の精度を保証するための実用的手法についても研究が進められている．本研究では，著者等が先行研究において開発した有限体積法における ICCG ソルバーをターゲットとし，実装方法，問題規模と低精度演算による性能改善の関係に注目し，5 種類のハードウェア環境下での検討を実施した．実装方法（疎

行列格納形式)，問題規模，ハードウェア環境によって，低精度演算を使用することにより，計算時間，消費電力(W)，消費エネルギー(J)への様々な効果があることがわかった．今後は同様な検討を他のアプリケーションについても実施する予定である．

本文中でも触れたが，P100，V100 では CPU の消費電力を考慮しておらず，アプリケーション全体としての消費電力特性の分析が必要である．

本実験で用いた OpenACC 版の実装では，CPU-GPU 間のデータ転送は GPU の持つ Unified Memory 機能に依存している．Unified Memory とは，本来別個である CPU と GPU のメモリ空間を同一のものと見せる技術であり，本実験で用いた GPU では，Page Migration Engine ハードウェアによるページ管理・転送機能がサポートされている．OpenACC の仕様において GPU の持つ Unified Memory 機能を利用するための指示文等は存在しないが，PGI compiler では拡張機能として提供され，コンパイル時にオプション -ta=tesla,managed を付与することで有効化される．この際 OpenACC の data 指示文によるユーザのデータ転送指示は無視され，ハードウェアにより CPU-GPU 間のデータの一貫性が保たれる．データ転送は CPU または GPU のいずれかでページフォルトが発生した際にページ単位で行われるため，小さなデータの転送は不利になると考えられ，またページフォルトに伴う割込み処理やドライバ等の処理がエネルギー効率に悪影響を及ぼす可能性がある．

特に本実装では収束判定を CPU で行っているため，リダクションの結果生じるスカラ要素を GPU から CPU に反復中何度も転送する必要がある．このリダクション要素の転送をページ単位で行う場合，性能に悪影響を及ぼすと考えられるが，OpenACC においてリダクション要素のデータ転送は隠蔽されており，data 指示文によりユーザが指示を行うことはできない．つまり，リダクション要素のデータ転送がどのように行われるかについては PGI compiler の実装依存である．NVIDIA Visual Profiler によるプロファイル結果等を見る限り，リダクション要素は例外的に Unified Memory を頼らない通常のデータ転送を行っているようではあるが，電力にどのような影響を及ぼすかについては未検証である．

謝辞

本研究の一部は，学際大規模情報基盤共同利用・共同研究拠点，および，革新的ハイパフォーマンス・コンピューティング・インフラ（JHPCN-HPCI)）の支援によるものです（課題番号: jh20037-NAH，課題名：高性能・変動精度・高信頼性数値解析手法とその応用)．また，本研究の一部は科学研究費補助金（19H05662，代表：中島研吾）の助成を受けたものである．

参考文献

1) 向け疎行列ソルバー，情報処理学会研究報告（2014-HPC-147-3)，2014
2) 中島研吾，大島聡史，塙敏博，星野哲也，伊田明弘，ICCG 法ソルバーの Intel Xeon Phi 向け最適化，情報処理学会研究報告（2016-HPC-157-16)，2016
3) 星野哲也，大島聡史，塙敏博，中島研吾，伊田明弘，OpenACC を用いたICCG法ソルバーのPascal GPUにおける性能評価，情報処理学会研究報告（2017-HPC-158-18)，2017
4) 坂本龍一，近藤正章，中島研吾，藤田航平，市村強，HPC アプリケーションにおける低精度演算の積極的利用による電力効率改善の検討，情報処理学会研究報告（2018-HPC-167-19)， 2018
5) Sakamoto, R., Kondo, M., Fujita, K., Ichimura, T., Nakajima, K., The Effectiveness of Low-Precision Floating Arithmetic on Numerical Codes: A Case Study on Power Consumption, ACM Proceedings of HPC Asia 2020, 2020
6) Mittal, A., A Survey of Techniques for Approximate Computing, ACM CSUR 48-4, 2016
7) 学際大規模情報基盤共同利用・共同研究拠点 2020 年度共同研究課題（課題番号: jh200037-NAH，課題名：高性能・変動精度・高信頼性数値解析手法とその応用)：
https://jhpcn-kyoten.itc.u-tokyo.ac.jp/abstract/jh200037-NAH
8) Washio, T., Maruyama, K., Osoda, T., Shimizu, F., 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
9) Nakajima, K., Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator, ACM/IEEE Proceedings of SC 2003, 2003
10) 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
11) Kreutzer, 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）C401‑C423
12) 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/
13) 最先端共同 HPC 基盤施設，http://jcahpc.jp/
14) STREAM Benchmark: https://www.cs.virginia.edu/stream/
15) RAPL: https://01.org/rapl-power-meter
16) Intel pcm-power.x: https://github.com/opcm/pcm