Vol.2020-HPC-175 No.2
2020/7/30

低精度・混合精度演算による高性能・高信頼性疎行列ソルバー

中島研吾[†1 †2] 荻田武史[†3] 塙敏博[†1] 河合直聡[†1] 伊田明弘[†1] 星野哲也[†1]

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には，メモリ・ネットワークの階層の深化に対応した通信最適化（Serial，Parallel）とともに省電力・省エネルギーに向けた検討が必要である．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，計算時間を短縮し省電力・省エネルギーを図る試みが活発に行われている．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が，実問題で現れる大規模疎行列・H 行列への応用例はほとんどない．本研究では，有限要素法，有限体積法から導出される大規模疎行列を係数行列とする連立一次方程式の求解に着目し，疎行列格納手法，問題規模と低精度演算による性能改善の関係に注目し，様々な計算機環境下での検討を実施した．また著者等によって提案された，M 行列性を有する疎行列を係数行列とする連立一次方程式向けの実用的な精度保証手法〔Ogita, Nakajima 2019〕を適用し，精度保証を実施した．

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〔1-7〕．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〔5,6〕．更に最新の研究〔7〕では，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 種類の計算機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，計算機環境によって，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．特に，問題規模が大きく，演算密度が場合には単精度演算による速度向上効果が顕著であることがわかった．本研究では，更に多様な計算機環境を想定した数値実験の他，半精度演算（FP16）の前処理への適用，精度保証を実施する．

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，計算手法，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〔8〕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 2018～2020 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）」〔9〕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〔4,6,7〕．当該研究では，これまであまり実施され

†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)
†3 東京女子大学現代教養学部
School of Arts and Sciences, Tokyo Woman's Christian University

てこなかった大規模疎行列向けの精度保証手法に関する研究開発を実施してきた．本稿では最新の研究成果に基づく手法〔10，11〕を適用し，検討を実施した．

3. 対象とするアプリケーション

3.1 概要

本研究では，著者等による先行研究〔1-7〕で使用されているプログラムに基づき検討を実施した．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，式（1）に示す定常熱伝導方程式を解くアプリケーション（P3D）を対象とし，導出された対称正定な疎行列を係数行列とする大規模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（ICCG）法によって解く〔1-7〕．

熱伝導率λの分布は図 2 に示すように，一層のみ$\lambda=\lambda_2$，他の部分は$\lambda=\lambda_1$とする．本節では$\lambda_2 \leqq \lambda_1$とし，$\lambda_1/\lambda_2$の比を様々に変化させた場合の計算を実施した．

$$\frac{\partial}{\partial x}\left(\lambda\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(\lambda\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(\lambda\frac{\partial T}{\partial z}\right)=f \qquad (1)$$

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

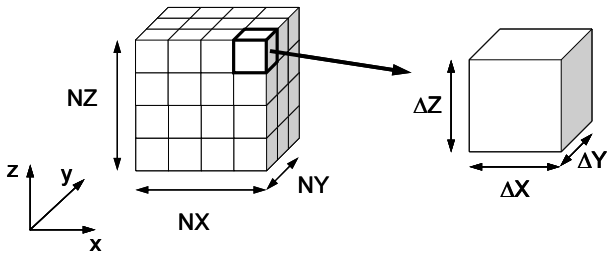

図 1　三次元ポアソン方程式ソルバーの解析対象
差分格子の各メッシュは直方体（辺長さはΔX，ΔY，ΔZ），
X，Y，Z 各方向のメッシュ数は NX，NY，NZ

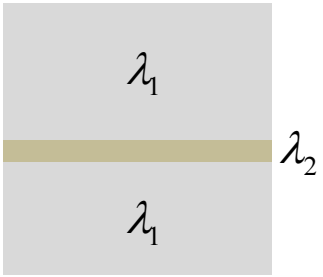

図 2　熱伝導率λの分布（$\lambda_2 \leqq \lambda_1$）

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〔1-7〕．

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（2）が得られる：

$$\left[\sum_{k=1}^{6}\frac{S_{ik}}{d_{ik}}\right]\phi_i-\left[\sum_{k=1}^{6}\frac{S_{ik}}{d_{ik}}\phi_k\right]=+V_i f_i \qquad (2)$$

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（2）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〔1-7〕．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〔1-7〕．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〔1-7〕．

オリジナルのプログラム〔1,2,3〕の実数変数は全て倍精度（FP64）であるが，本研究では，これを全て単精度（FP32）にした場合，部分的に単精度演算，半精度演算（FP16）を導入した混合精度演算の効果について検討する．

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〔1-7〕．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．

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〔1-7,12,13〕．図 3 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．

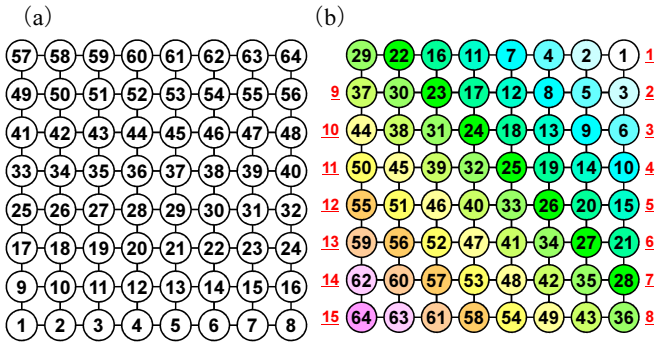

図 3　CM-RCM(k)法による色づけとリオーダリング，(a) 元のグラフ，(b) RCM 法によるリオーダリング（赤字はレベルセット番号），(c) CM-RCM(k)法による再リオーダリング (4 色：CM-RCM(4))，各色内の要素数は 16 でバランス

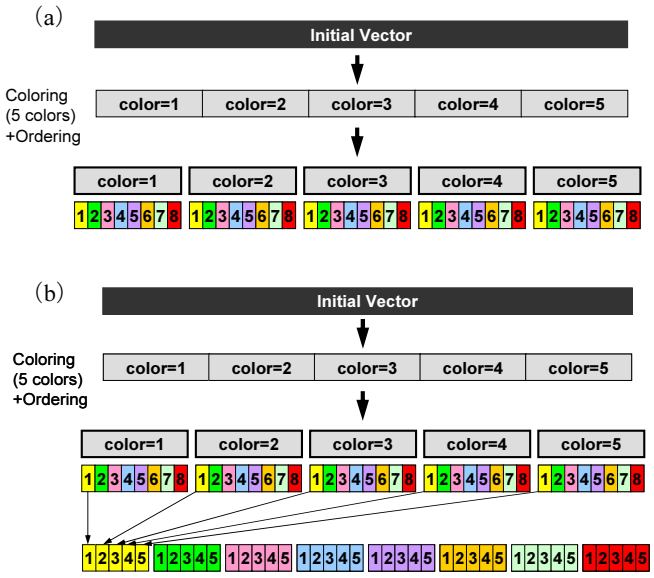

図 4　要素の番号付け（a）CM-RCM(k)法による番号付け（Coalesced Numbering），（b）Sequential Reordering による再番号付け（各スレッド上の要素は連続な番号付け）

3.3 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 4 (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 4 (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〔1-7〕，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〔1-7〕．

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 5（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．

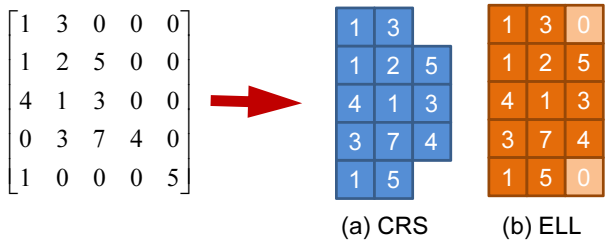

図 5　疎行列の格納形式（a）CRS，（b）ELL

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〔1-7,14〕．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 6 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

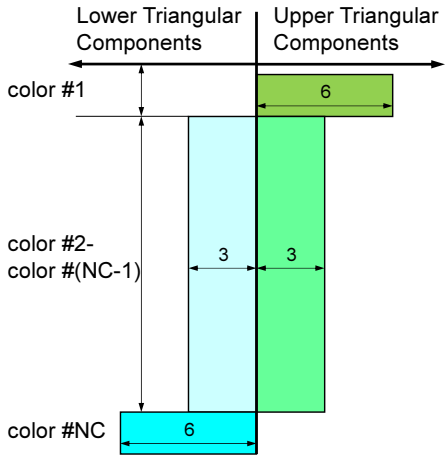

図 6　CM-RCM(k)法における上下三角成分数（NC：総色数）

著者等の先行研究〔1,2,7〕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 7）．図 6 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 8 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 7 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．

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

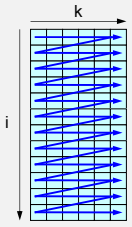

図 7　ELL 形式の前進代入への適用例（Row-wise），非零非対角成分の最大数=6．NCOLORtot：総色数，PEsmpTOT：総スレッド数，Index(ip,icol)：各色，スレッドに属する要素総数，AML(k,i)：非零非対角成分，IAML(k,i)：非零非対角成分（列番号），DD(i)：対角成分．

ただしこのような手法は，やや非効率的であり，図 8 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〔15〕が提案されている（図 8）．

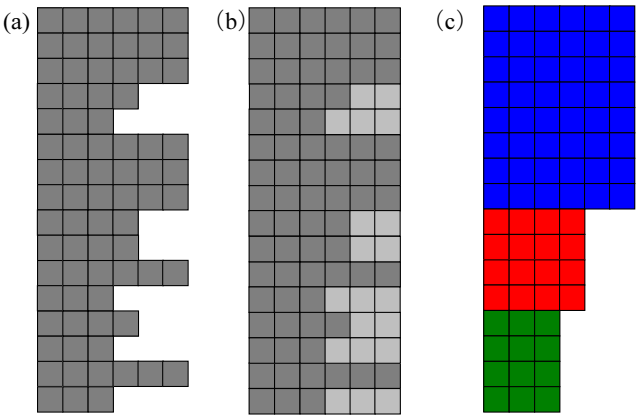

図 8　ELL 形式の不規則行列への適用例 (a) CRS，(b) ELL，(c) Sliced-ELL

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 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〔13〕．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．

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

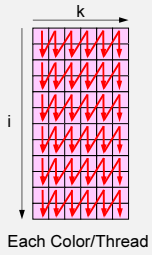

図 9　ELL 形式の前進代入への適用例（Column-wise），非零非対角成分の最大数は 6 としてある．NCOLORtot：総色数，PEsmpTOT：総スレッド数，Index(ip,icol)：各色，スレッドに属する要素総数，AML(i,k)：非零非対角成分，IAML(i,k)：非零非対角成分（列番号），DD(i)：対角成分．各色，スレッドに対応したブロックにおいて計算が実施される（本図では各ブロックのサイズは 2 としてある）．

図 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．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．

3.6 SELL-C-σ

SELL-C-σ〔16〕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．

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（=512 bit/64））に設定するとベクトル化の効率が高まる. 本研究では, 3.4 で述べた Row-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.

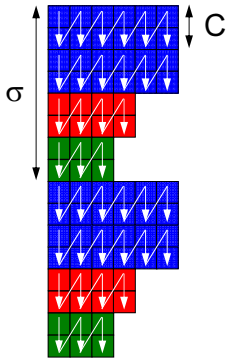

図 10　SELL-C-σ形式〔16〕

```
!$omp parallel (…)
    do ic= 2, NCOLORtot-1
!$omp do
      do ip= 1, PEsmpTOT
        iq1= SMPindex((ip-1)*NCOLORtot + ic-1) + 1
        iq2= SMPindex((ip-1)*NCOLORtot + ic)
        ip0= (ip-1)*NCOLORtot + ic
        iq0= (iq1-1)*8
      do ib  = iq1, iq2
        ib0= (ib-iq1)*8
!$omp simd
      do is  = 1, 8
        i= iq0 + ib0 + is
        VAL= W(i,Z)
        do k= 1, 3
          VAL= VAL- AL(ib0+is,k,ip0)*W(itemL(ib0+is,k,ip0),Z)
        enddo
        W(i,Z)= VAL * W(i,DD)
      enddo
      enddo
      enddo
    enddo
!$omp end parallel
```

図 11　SELL-8-1 形式の前進代入への適用例（Row-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，SMPindex (0:NCOLORtot×PEsmpTOT)：各色，スレッドに属する要素インデックス

3.7 OpenMP オーバーヘッド軽減の更なる試み（Barrier-Free 実装）

図 12 に示すように前処理付き共役勾配法(Preconditioned Conjugate Gradient，PCG）は，様々なループ処理（行列ベクトル積，内積，AXPY（ベクトル定数倍の加減），前処理（ICCG の場合は前進後退代入））から構成されている．各ステートメントに!$omp parallel を適用すると，各ステートメント毎に fork-join が発生するためオーバーヘッドが発生する.

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    solve [M]z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
    else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

図 12　前処理付き共役勾配法（Preconditioned Conjugate Gradient，PCG）のアルゴリズムの概要

著者等の先行研究〔1,2〕では，OpenMP によるオーバーヘッドを極力削減するための検討がなされているが，〔3〕では更なるオーバーヘッド削減のために, 図 13 に示すような変更を加えた「Barrier-Free 実装」を提案し，大幅な性能向上が得られている〔3,7〕.「Barrier-Free 実装」の特徴は以下の通りである：

- 反復法のループの前後にのみ「!$omp parallel，!$omp end parallel」を挿入する（fork-join が一回しか起こらない）
- 「!$omp do」,「reduction」を除去

```
Compute r(0)= b-[A]x(0)
!$omp parallel (…)
for i= 1, 2, …
    solve [M]z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
    else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
End
!$omp end parallel
```

図 13　「Barrier-Free 実装」に基づく前処理付き共役勾配法（Preconditioned Conjugate Gradient，PCG）のアルゴリズムの概要〔3〕

図 14 は図 11 に示す SELL-8-1 による前進代入部に「Barrier-Free 実装」を適用した例である．「!$omp parallel，!$omp end parallel」による fork-join のオーバーヘッドが生じるのは反復法ループの前後一回だけとなり，「!$omp do」も除去されている.

図 15 は内積（$\rho_{i-1}=r^{(i-1)}z^{(i-1)}$），AXPY（ベクトル定数

倍加減，$p^{(i)}= z^{(i-1)}+\beta_{i-1}p^{(i-1)}$）への適用事例である．Barrier-Free 実装により，「reduction」が除去されていることがわかる．

```
!$omp parallel (…)
(…)
    do ic= 2, NCOLORtot-1


      iq1= SMPindex((ip-1)*NCOLORtot + ic-1) + 1
      iq2= SMPindex((ip-1)*NCOLORtot + ic)
      ip0= (ip-1)*NCOLORtot + ic
      iq0= (iq1-1)*8
      do ib  = iq1, iq2
        ib0= (ib-iq1)*8
!$omp simd
      do is  = 1, 8
        i= iq0 + ib0 + is
        VAL= W(i,Z)
        do k= 1, 3
          VAL= VAL- AL(ib0+is,k,ip0)*W(itemL(ib0+is,k,ip0),Z)
        enddo
        W(i,Z)= VAL * W(i,DD)
      enddo
      enddo
!$omp barrier
    enddo
(…)
!$omp end parallel
```

図 14　「Barrier-Free 実装」を図 11 に示す SELL-8-1 形式の前進代入へ適用した例（Row-wise，Sequential）

(a)

```
      RHO= 0.d0
!$omp parallel do private(i) reduction(+:RHO)
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

      if ( L.eq.1 ) then
!$omp parallel do private(i)
        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
      else
        BETA= RHO / RHO1
!$omp parallel do private(i)
        do i= 1, N
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
      endif
```

(b)

```
      W_RHO(ip)= 0.0d0
!$omp simd
      do i= (ip-1)*ls+1, min(ip*ls,N)
        W_RHO(ip)= W_RHO(ip) + W(i,R)*W(i,Z)
      enddo
      RHO= 0.d0
!$omp barrier
!$omp simd
      do i = 1, PEsmpTOT
        RHO= RHO + W_RHO(i)
      end do

      if ( L.eq.1 ) then
        do i= (ip-1)*ls+1, min(ip*ls,N)
          W(i,P)= W(i,Z)
        enddo
!$omp barrier
      else
        BETA= RHO / RHO1
        do i= (ip-1)*ls+1, min(ip*ls,N)
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
!$omp barrier
      endif
```

図 15　内積（$\rho_{i-1}=r^{(i-1)}z^{(i-1)}$），AXPY（ベクトル定数倍加減，$p^{(i)}= z^{(i-1)}+\beta_{i-1}p^{(i-1)}$）の並列化事例（a）オリジナル実装，（b）「Barrier-Free 実装」

3.8　予備的計算事例（倍精度・単精度の比較）

図 2 において，λ_1/λ_2 を変化させ，倍精度，単精度により計算を実施した例を示す．Reedbush-U システム（東京大学情報基盤センター，Intel Xeon E5-2695 v4　(Broadwell-EP)〔4,5,7,17〕の 1 ノード（36 コア）を使用して計算を実施した．疎行列格納法として CRS（Compressed Row Storage）を適用し，番号付けを Sequential とした場合の計算を実施する〔10〕．問題サイズは NX=NY=NZ=128（N=2,097,152）とし，CM-RCM（20）（色数＝20 の CM-RCM 法）を適用した〔4,5〕．図 16 は，倍精度演算時の反復回数，計算時間を 1 とした場合の単精度演算時の反復回数，計算時間の比である．

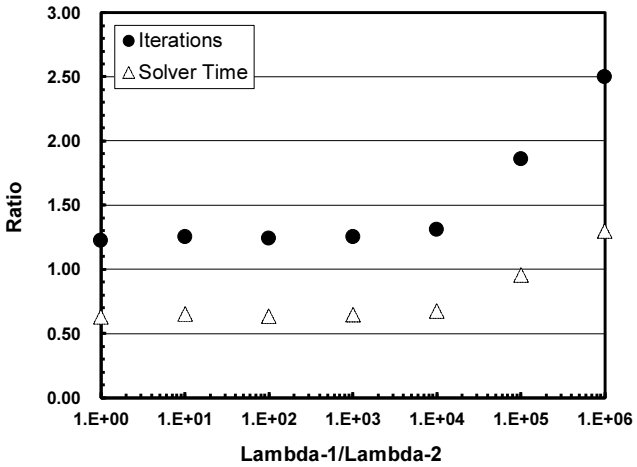

図 16　単精度／倍精度演算時の反復回数比・計算時間と λ_1/λ_2 の関係（NX=NY=NZ=128）（倍精度の場合を 1 とする）（東大 Reedbush-U 1 ノード（36 コア））●：反復回数，△：計算時間

λ_1/λ_2（条件数に比例）が大きくなるに従って単精度演算時の反復回数が相対に大きくなっていることがわかる．単精度演算の反復回数が安定している$\lambda_1/\lambda_2=10^4$までは，計算時間の単精度／倍精度比は 0.60 程度であるが，それより大きい場合は，単精度演算の反復回数が増加して，倍精度演算の場合より計算時間が増えている．図 17 は，図 1，図 2 における，Bottom 面と Top 面の代表点の計算結果の誤差である．倍精度演算の結果が正しいものとして，単精度演算結果との誤差を示している．$\lambda_1/\lambda_2=10^6$では誤差が約 10 に達し，正しい解から大きな差がある．

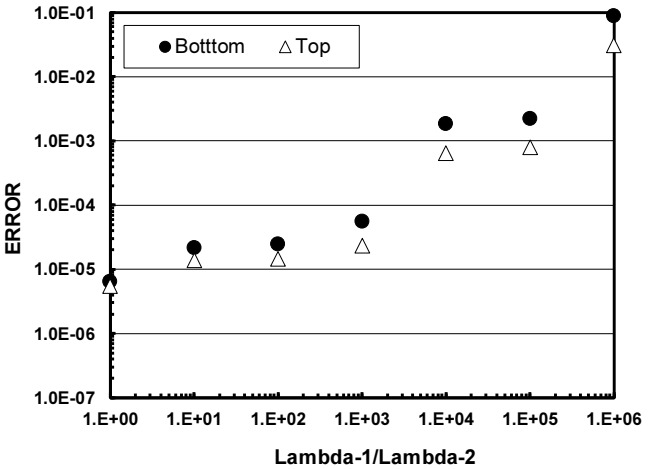

図 17　単精度／倍精度演算時の代表点（Bottom・Top 面）（図 1）計算結果相対誤差と λ_1/λ_2 の関係（(NX=NY=NZ=128）

4. 精度保証について

2．で照会した学際大規模情報基盤共同利用・共同研究拠点 2018～2020 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）」〔9〕では実問題に精度保証を適用するための基盤の研究開発を実施しており，2019 年度は本研究で扱う三次元熱伝導問題の数値解の精度保証を実施した．

偏微分方程式の数値解法では，有限要素法，有限体積法，差分法等によって時間・空間方向の離散化を実施し，得られた連立一次方程式 $Ax=b$ を解くことが多いが，離散化誤差については，独立に考えることができるため，本研究では，連立一次方程式求解に対して精度保証を適用する．

係数行列 A が M 行列性（全ての非対角成分が 0 以下であり，全ての固有値の実部が正である行列を M 行列と言う〔10，11〕）を持つ場合，高速な精度保証手法〔18〕の適用が可能である．この手法を使用して，精度評価を実施したところ，$\lambda_1/\lambda_2=10^6$ の場合，倍精度演算でも最大相対誤差が 10^0 のオーダーになってしまい，やや厳しすぎる判定であり，より精密に誤差の最大値を推定する手法の研究開発が必要であることがわかった．

そこで，それを抑制するための方法〔19〕を適用した．具体的には，以下のようなアルゴリズムとなる〔10,11〕．

① 離散化によって得られた連立一次方程式 $Ax=b$ を解き，得られた近似解を $\hat{x}$ とする
② 連立一次方程式 $Ay=e$（e はすべての要素が 1 のベクトル）を解く．得られた近似解を $\hat{y}$ とする
③ ②の $\hat{y}$ を用いて係数行列 A の M 行列性の保証を行う
④ 残差 $r=b-A\hat{x}$ を精度保証付きで計算し，残差の近似値を $\hat{r}$，誤差限界を e_r とする．
⑤ 連立一次方程式 $Az=\hat{r}$ を解き，近似解を $\hat{z}$ とする．
以下の誤差評価式の右辺について，浮動小数点演算の丸めモードの変更を適宜使用しながら，厳密な上限を求める（右辺第 2 項の分母が 0 以下になった場合は精度保証失敗）：

$$\|x-\hat{x}\|_\infty \le \|\hat{z}\|_\infty + \frac{\|\hat{y}\|_\infty \|b-A(\hat{x}+\hat{z})\|_\infty}{1-\|e-A\hat{y}\|_\infty} \qquad (3)$$

3.8 でも使用した Reedbush-U システム（東京大学情報基盤センター，Intel Xeon E5-2695 v4 (Broadwell-EP)）の 1 ノード（36 コア）を使用した評価を実施した．問題設定等は 3.8 の場合と全く同じである．

ICCG 法の反復の停止条件は残差ノルムを用いて：

$$\begin{aligned} &Ax=b, \|b-A\hat{x}\|_2/\|b\|_2 < \varepsilon_1 \\ &Ay=e, \|e-A\hat{y}\|_\infty < \varepsilon_2 \\ &Az=\hat{r}, \|\hat{r}-A\hat{z}\|_2/\|\hat{r}\|_2 < \varepsilon_3 \end{aligned} \qquad (4)$$

のように与えた．但し，それぞれの残差ベクトルの計算については ICCG 法の反復中に計算される見かけの残差ベクトルを用いた．また，浮動小数点演算は，すべて倍精度を用いた．また，ここでは，$\varepsilon_1=10^{-12}$，$\varepsilon_2=10^{-2}$，$\varepsilon_3=10^{-9}$ とした．

離散化して得られた連立一次方程式の近似解を計算し，精度保証法を用いて，最大相対誤差の上限（式（5））及び相対残差ノルム（式（6））を求めた：

$$\max_{1\le i\le n}\left|\frac{x_i-\hat{x}_i}{x_i}\right| \qquad (5)$$

$$\|b-A\hat{x}\|_2/\|b\|_2 \qquad (6)$$

の上限を求めた．結果を図 18 に示す．

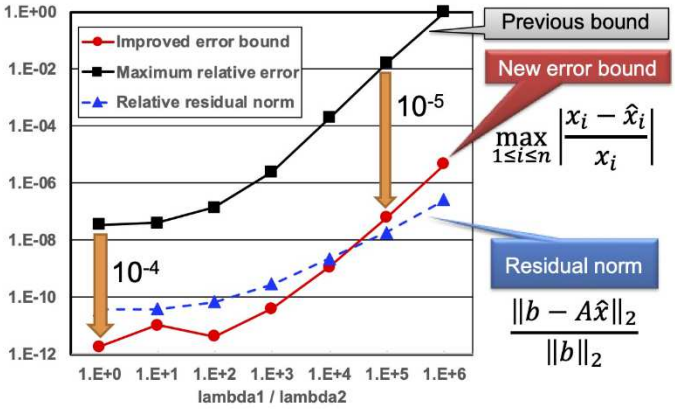

図 18 条件（λ_1/λ_2）毎の最大相対誤差（実線）と相対残差ノルム（破線）〔10,11,19〕，黒い実線（Previous bound）は〔18〕に基づく厳しめの誤差上限

図 18 から，提案方式によって誤差限界の過大評価（黒い実線，Previous bound）が大幅に抑制できていることがわかる．本研究において得られた連立一次方程式に対しては，実用的かつ高精度な精度保証法の適用に成功した．本研究ではこの他，単精度演算等の低精度演算での計算に関する適用を実施する．

5. 計算機環境

本研究では以下の 4 種類の計算機環境（マルチコア・メニィコア CPU）を使用した：

① Oakforest-PACS（OFP）（最先端共同 HPC 基盤施設）〔20〕
② Oakbridge-CX（OBCX）（東京大学情報基盤センター）〔17〕
③ AMD Rome（Rome）（日本 AMD より貸与）〔21〕
④ Oakleaf-7（OL7）（東京大学情報基盤センター）

本研究では，各 1 ノードを使用して実施した．表 1 は①～④の各 1 ソケットの概要である．

AMD Rome（Rome）では，Intel コンパイラで静的にコンパイルした実行形式を使用した．Oakleaf-7 は東京大学情報基盤センターに導入されている Fujitsu PRIMEHPC FX700〔22〕であり，A64FX を 8 基搭載しているが，今回はそのうちの 1 基（1 ノード）を使用した．

本研究で対象とする P3D アプリケーションでは疎行列を係数行列とする連立一次方程式の求解部分（ICCG 法）にほとんどの計算時間が費やされる．この部分は Memory-Bound であるため，メモリ性能が計算時間に大きく影響する．

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	Oakforet-PACS	Oakbridge-CX	AMD Rome	Oakleaf-7（FX700）
略称	OFP	OBCX	Rome	OL7
CPU 名称	Intel Xeon Phi 7250 (Knights Landing, KNL)	Intel Xeon Platinum 8280 (Cascade Lake, CLX)	AMD EPYC 7742 (Rome)	Fujitsu A64FX (1.8GHz)
コア数/ソケット	68	28	64	48
ソケット数/ノード	1	2	2	1
理論演算性能（GFLOPS）	3,046	2,419	2,304	2,765
主記憶容量（GB）	MCDRAM: 16 DDR4: 96	96	256	32
メモリ性能（GB/sec）STREAM Triad [23])	MCDRAM: 490 DDR4: 84.5	101	177	809
コンパイラ	Intel Parallel Studio 2019			Fujitsu

6. 計算例

6.1 実施ケースの概要

本研究では，以下の 5 種類の行列格納形式について検討した：

① Compressed Row Storage（CRS）（図 8）
② Row-Wise な Sliced ELL（ELL）（図 7，8）〔7,15〕
③ ELL に「Barrier-Free 実装」（図 13-15〔3,7〕）を適用した手法（ELLo）
④ SELL-C-σ（C=8, σ=1）（SCS）（図 10，11）〔7,16〕
⑤ SCS に「Barrier-Free 実装」（図 13-15〔3,7〕）を適用した手法（(OPT）

OBCX については 1 ノード（2 ソケット），合計 56 コアのうち 48 コア（48 スレッド）を使用した．また OFP では，先行研究〔1,2,7〕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．Rome は 128 コア（128 スレッド），OL7 は 32 コア（32 スレッド）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 を使用することができる．Cache モードでは，MCDRAM を DDR4 のキャッシュとして使用できる他，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，Flat モードで MCDRAM のみ使用して実施した．

問題規模としては，先行研究〔7〕において，単精度・倍精度，疎行列実装方法による計算性能の差異が比較的現れやすい 2,097,152 DOF（$=128^3$）に固定して実施し，CM-RCM(10)を適用した．

計算は大きくわけて表 2 に示す 3 つの項目について実施した．計算は全て 5 回実施し，最短の時間を計算時間として採用した．

表 2　本研究における計算項目

設定	概　要	疎行列格納形式	λ_1/λ_2	ε_I（式(4)）
A	行列格納方法	CRS，ELL，ELLo，SCS，OPT	10^0	10^{-8}
B	混合精度演算	CRS	10^0~10^6	10^{-8}
C	精度保証			10^{-12}

6.2 行列格納方法の比較（A）

CRS，ELL，ELLo，SCS，OPT の各格納形式において，ICCG 法計算部分の計算時間の評価を実施した．ここでは，表 2 に示すように，$\lambda_1=\lambda_2=1.00$ の場合のみ実施した．倍精度演算を適用し（CRS-D, ELL-D, ELLo-D, SCS-D, OPT-D），OPT のみ単精度演算を適用した事例を実施した（OPT-S）．

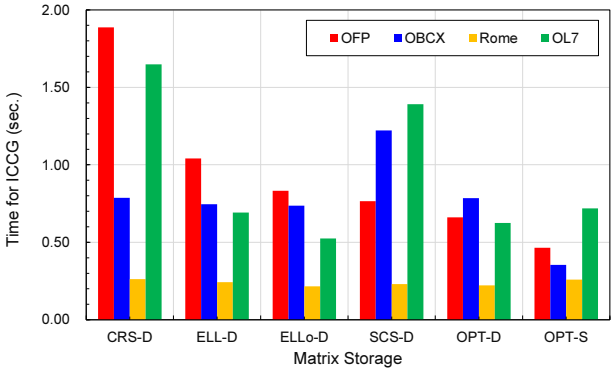

図 19　疎行列格納法による ICCG 法計算時間の比較

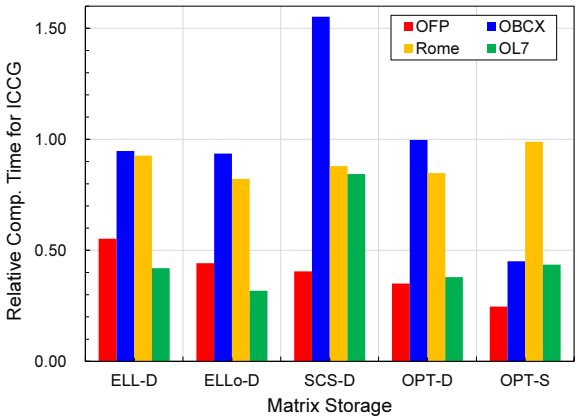

図 20　疎行列格納法による ICCG 法計算時間の比較（CRS-D の計算時間を 1.00 とした場合，1.00 より小さいと CRS-D より速い）

図 19，図 20 はそれぞれ各格納手法について，各計算機環境における ICCG 法の計算時間を測定したものである．図 19 は実時間，図 20 は各計算機において CRS-D の計算時間を 1.00 として無次元化したものである．

OFP と OBCX については，ELLo を除いては既に〔7〕で紹介した結果と同じである．OFP は CRS⇒ELL⇒ELLo⇒SCS⇒OPT と演算密度が増すにつれて計算効率が改善されており，OPT-D⇒OPT-S（倍精度⇒単精度）では反復回数が 20%ほど増加するものの，計算時間は 30%程度減少している．OBCX では，ELL/ELLo⇒SCS で一旦計算効率は低下するが，OPT-D⇒OPT-S（倍精度⇒単精度）は計算時間は 55%程度減少している．ELL⇒ELLo の計算効率改善は OFP においてより顕著に表れている．

Rome と OBCX を比較すると，ピーク演算性能はほぼ同じ，STREAM Triad メモリバンド幅は Rome が約 1.7 倍，コア当たりのキャッシュ容量は約 3 倍である（OBCX：1.375 MB/core，Rome：4.000 MB/core）．また OBCX の L3 キャッシュがソケット（28 コア）で共有されているのに対して，Rome では 4 コアごとに 16MB 共有という違いがあるが〔21〕，本研究における ICCG 法の計算性能は，Rome の方が 3 倍～4 倍高速である．他の疎行列ソルバーではほぼメモリバンド幅に比例する性能が得られているため，本ケースにおいて特に Rome が速くなっている理由は不明である．Rome は疎行列格納形式による計算性能の差異はほとんど見られず，また，OPT-D⇒OPT-S（倍精度⇒単精度）では，1 反復当たりの計算効率は「倍精度⇒単精度」で 5.5%ほど改善が見られるものの，反復回数が 20%程度増えたことにより却って全体の計算時間が長くなっている．

OL7 は OFP と同様に CRS⇒ELL⇒ELLo については顕著な性能向上が見られ，特に ELL⇒ELLo では 30%以上の速度向上が得られている．SCS では OBCX を同様に遅くなるが，OPT によって 120%程度の速度向上が得られている．また，OPT-D⇒OPT-S では，1 反復あたりの計算効率は 7%程度改善しているものの，ICCG 法としての計算時間は Rome 同様に OPT-S は OPT-D より長い．ELL⇒ELLo，SCS⇒OPT における性能改善から明かなように，3.7 で示した「Barrier-Free 実装」の効果は顕著である．現在，OL7 では Fujitsu 製コンパイラを使用しているが，現状では SCS, OPT で図 11，図 14 に示すように「!$omp simd」を挿入した状態では計算性能が大幅に低下し，10 倍程度の計算時間がかかるため，SCS，OPT では「!$omp simd」を除いて測定を実施している．したがって現状では充分にベクトル化，SIMD 化が効いていない状況である．〔7〕でも明らかになったように，ベクトル化，SIMD 化が充分でなく，計算密度が小さい場合には倍精度⇒単精度による速度向上が充分に得られていないものと考えられる．

6.3 混合精度演算の効果（B）

P3D アプリケーションを対象とした先行研究〔4,5,6,7〕ではプログラムの全変数を倍精度または単精度とした場合について比較を実施してきた．前処理付き反復法では，前処理部分のみ他の部分と演算精度を変える，混合精度演算が古くから行われている〔24〕．ここでは，前処理部分（前進後退代入）について低精度演算を適用した例を紹介する．実施例は，表 3 及び図 21（a-c）に示す通りである．OL7 では半精度（FP16）を手軽に実行できるが，半精度は有効桁が 3 桁程度であるため，ベクトルを半精度とすると解が発散する．ここでは係数行列のみ半精度とし，ベクトルは単精度として扱った（図 21（b,c））．疎行列格納形式は CRS である．

表 3　混合精度演算の効果

	反復法 主要部	前処理	前処理部分 ベクトル	
D-D	倍精度	倍精度	倍精度	
D-S	倍精度	単精度	単精度	図 21（a）
D-H	倍精度	半精度	単精度	図 21（b），OL7 のみ
S-S	単精度	単精度	単精度	
S-H	単精度	半精度	単精度	図 21（c），OL7 のみ

(a)

```
!$omp parallel do private(ip,i)
      do ip= 1, PEsmpTOT
      do i= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
        Ws(I,Z)= W(I,R)
      enddo
      enddo

!$omp parallel private(ic,ip,ip1,I,WVALs,k)
      do ic= 1, NCOLORtot
!$omp do
        do ip= 1, PEsmpTOT
          ip1= (ip-1)*NCOLORtot + ic
        do i= SMPindex(ip1-1)+1, SMPindex(ip1)
          WVALs= Ws(i,Z)
          do k= indexL(i-1)+1, indexL(i)
            WVALs= WVALs -  ALs(k) * Ws(itemL(k),Z)
          enddo
          Ws(i,Z)= WVALs * Ws(i,DD)
        enddo
        enddo
      enddo
!$omp end parallel

(Backward Substitution)

!$omp parallel do private(ip,i)
      do ip= 1, PEsmpTOT
      do i= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
        W(I,Z)= Ws(I,Z)
      enddo
      enddo
```

(b)

```
!$omp parallel do private(ip,i)
      do ip= 1, PEsmpTOT
      do i= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
        Ws(i,Z)= W(i,R)
      enddo
      enddo

!$omp parallel private(ic,ip,ip1,i,WVALs,k)
      do ic= 1, NCOLORtot
!$omp do
        do ip= 1, PEsmpTOT
          ip1= (ip-1)*NCOLORtot + ic
        do i= SMPindex(ip1-1)+1, SMPindex(ip1)
          WVALs= Ws(i,Z)
          do k= indexL(i-1)+1, indexL(i)
            WVALs= WVALs -  ALh(k) * Ws(itemL(k),Z)
          enddo
          Ws(i,Z)= WVALs * Wh(i,DD)
        enddo
        enddo
      enddo
!$omp end parallel

(Backward Substitution)

!$omp parallel do private(ip,i)
      do ip= 1, PEsmpTOT
      do i= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
        W(I,Z)= Ws(I,Z)
      enddo
      enddo
```

(c)

```
!$omp parallel do private(ip,i)
      do ip= 1, PEsmpTOT
      do i= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
        Ws(i,Z)= Ws(i,R)
      enddo
      enddo

!$omp parallel private(ic,ip,ip1,i,WVALs,k)
      do ic= 1, NCOLORtot
!$omp do
        do ip= 1, PEsmpTOT
          ip1= (ip-1)*NCOLORtot + ic
        do i= SMPindex(ip1-1)+1, SMPindex(ip1)
          WVALs= Ws(i,Z)
          do k= indexL(i-1)+1, indexL(i)
            WVALs= WVALs - ALh(k) * Ws(itemL(k),Z)
          enddo
          Ws(i,Z)= WVALs * Wh(i,DD)
        enddo
        enddo
      enddo
!$omp end parallel
```

図 21　混合精度演算（前進代入部分）(a) 倍精度＋単精度前処理（D-S），(b) 倍精度＋半精度前処理（D-H），(c) 単精度＋半精度前処理（S-H），**緑字：倍精度**，**赤字：単精度**，**青字：半精度**

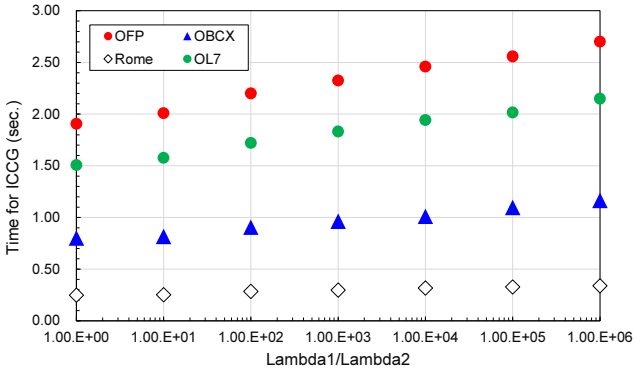

図 22　ICCG 法の計算時間（倍精度＋倍精度前処理（D-D），CRS），λ_1/λ_2 の影響

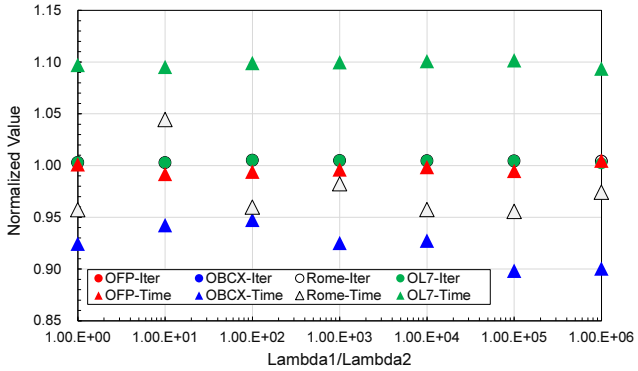

図 23　ICCG 法の反復回数・計算時間（倍精度＋単精度前処理（D-S），CRS），λ_1/λ_2 の影響，（D-D）の反復回数・計算時間で無次元化

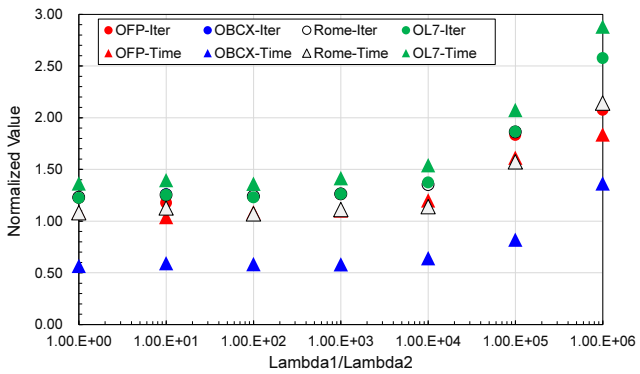

図 24　ICCG 法の反復回数・計算時間（単精度＋単精度前処理（S-S），CRS），λ_1/λ_2 の影響，（D-D）の反復回数・計算時間で無次元化

図 22 は，倍精度＋倍精度前処理（D-D）の場合の，各（λ_1/λ_2）における ICCG 法の計算時間である．傾向としては図 16，図 18 と同様であり，λ_1/λ_2 が増加すると条件数が大きくなり反復回数が増加し，計算時間もそれに比例して増加する傾向にある．D-D の場合は，反復回数は各計算機環境において変わらないため，計算時間は図 19 に示した CRS-D の場合に比例している．

図 23 は倍精度＋単精度前処理（D-S）の場合の ICCG 法の反復回数，計算時間を D-D の場合の値で無次元化したものである．反復回数は単精度前処理導入によって変化がない．計算時間については，OBCX が最大 10%程度減少している他，OFP はほぼ変化無し，Rome が最大 5%程度減少している他，OL7 では 10%程度計算時間は増加している．前進後退代入部分は ICCG 法の計算時間の 45～50%を占めているが，倍精度⇒単精度⇒倍精度のコピーや同期オーバーヘッドの影響もあり効果は計算性能面での効果は大きくないが，λ_1/λ_2 が 10^6 の場合でも反復回数は D-D の場合と変化せず，正しい答えが得られている．

図 24 は単精度＋単精度前処理（S-S）の場合であり，基本的には図 16 に対応している．反復回数は計算機環境によって変わらないが若干の変動があり，$\lambda_1/\lambda_2=10^6$ の場合の OL7 は他環境より反復回数が多い．計算時間も反復回数に比例している．OBCX が λ_1/λ_2 が 10^4 以下の場合に計算時間が D-D の約 50%程度である他，OFP，Rome，OL7 では D-D と同様か増加している．図 25 は，反復当たりの計算時間の単精度（S-S）と倍精度（D-D）の比である．OBCX は 46%程度であるのに対して，OFP，Rome では 88%程度，OL7 では逆に約 110%強と単精度の方が一反復当たりの計算時間は増加している．

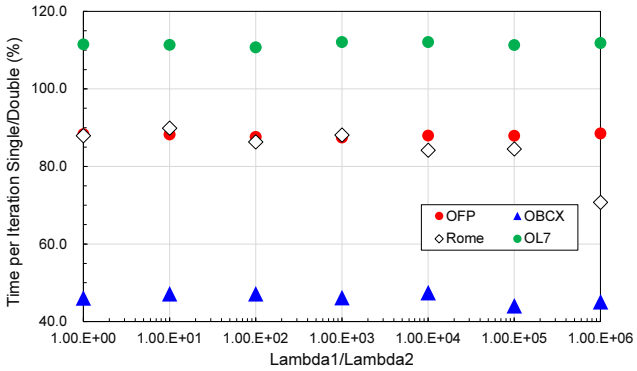

図 25　ICCG 法の 1 反復あたりの計算時間の比，単精度（S-S）の計算時間を倍精度（D-D）の計算時間で無次元化

図 26～27 は，OL7 において半精度演算を導入した場合の効果を示したものである．図 26 は図 1・図 17 における Bottom の点における計算結果を倍精度＋倍精度前処理（D-D）の場合の解に対する相対誤差として計算したものである．前処理に半精度演算を適用した D-H，S-H は $\lambda_1/\lambda_2=10^6$ の場合は収束解が得られていない他は，D-S，D-H

は D-D と比較して計算誤差は無い．一方で S-S，S-H は $\lambda_1/\lambda_2=10^3$ の以下の場合は誤差も 1%以下であるが，λ_1/λ_2 が増加すると急速に誤差が増加している．

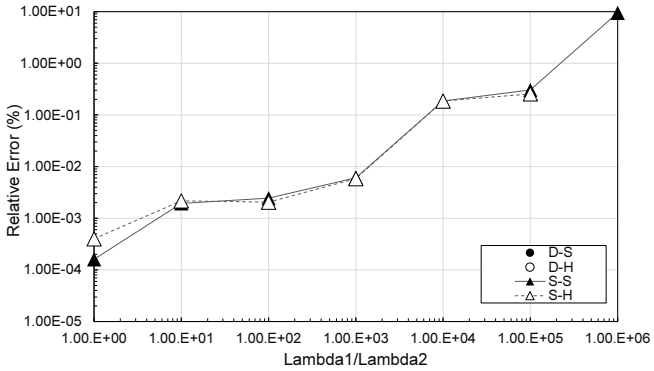

図 26　Bottom 点（図 1・図 17）での（D-D）の結果との相対誤差，D-S，D-H は解が得られる範囲では誤差は無い

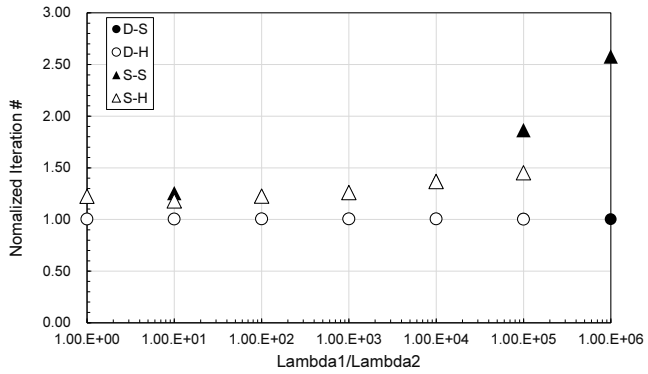

図 27　各ケースにおける反復回数（D-D の反復回数で無次元化）

図 27 は，各ケースにおける反復回数であり，D-S，D-H では解が得られる範囲では，反復回数は D-D の場合と変化していないことがわかる．S-H は S-S とほぼ同様である．

6.4 精度保証（C）

4．で紹介した精度保証手法〔10,11〕を P3D アプリケーションに適用した事例を紹介する．表 4 に示す 3 つの場合ついて検討を実施した．

表 4　精度保証計算項目

	P3D 本体	精度保証部
D-D	倍精度	倍精度
D-S	倍精度	単精度
S-S	単精度	単精度

図 28 は OBCX において，$\lambda_1/\lambda_2=10^3$ のを変化させた場合の，表 4 の D-D における，本体 ICCG 法求解時間（Solver）と精度保証部（Verification）の計算時間の内訳である．反復法の打ち切り誤差 $\varepsilon_1=10^{-12}$ となっているため，ICCG 法部分に従来よりやや計算時間がかかっている．精度保証部の計算時間は ICCG 法求解時間と比較してやや長い．

図 29 は表 4 に示す各ケースにおいて，λ_1/λ_2 を変化させた場合の計算時間（ICCG 法求解＋精度保証）である．$\lambda_1/\lambda_2=10^4, 10^5, 10^6$ の場合は精度保証に失敗している．

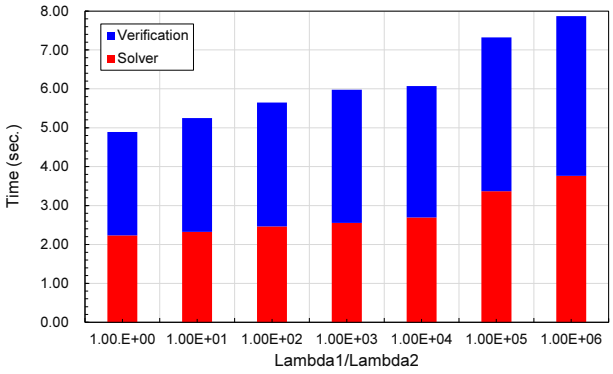

図 28　OBCX 上での計算時間の内訳（■：Solver（ICCG 法求解），■：Verification（精度保証部））（D-D（表 4））

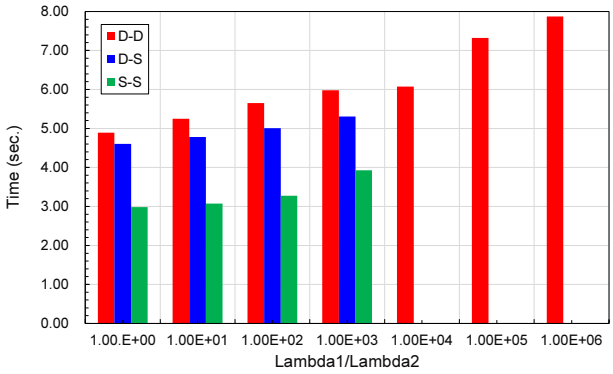

図 29　OBCX 上の計算時間（ICCG 法求解＋精度保証）

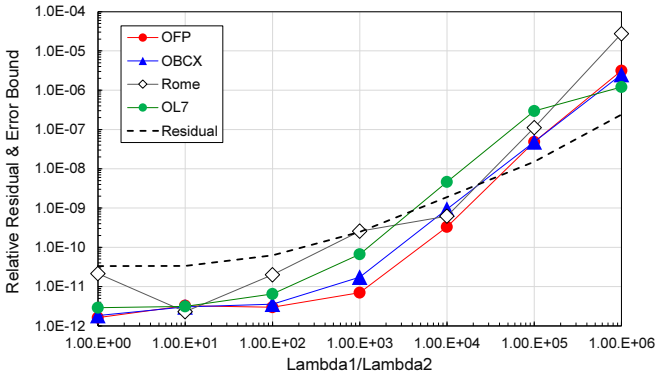

図 30 P3D アプリケーションの精度保証結果（D-D），相対残差ノルム（点線，OBCX における結果）と各計算機環境における相対最大誤差

図 30 は，P3D アプリケーションの精度保証結果（D-D）であり相対残差ノルムと各計算機環境における相対最大誤差を示している．相対残差ノルムと相対最大誤差はほぼ同じオーダーであり，先行研究〔10,11〕の結果を確認することができた．図 30 に示す相対残差ノルム（$\|b-A\tilde{x}\|_2/\|b\|_2$）は OBCX における値である．倍精度演算においては計算機環境によって収束までの ICCG 法の反復回数は同じであるが，相対残差ノルムの値は若干異なっており，求められた解も同様に異なっているが，ほぼ一致していると考えられる．ただ，相対最大誤差は図 30 に示すように，計算機環境によ

って異なっており，最大1桁程度の差がある．図31はOBCXにおける相対最大誤差である．D-D と D-S の相対最大誤差は精度保証が成功している範囲では等しい．図 29 に示すように， D-S は D-D と比較して短時間で求解＋精度保証のプロセスを実施することが可能であるため，比較的条件の良い問題では，有効に活用できる．

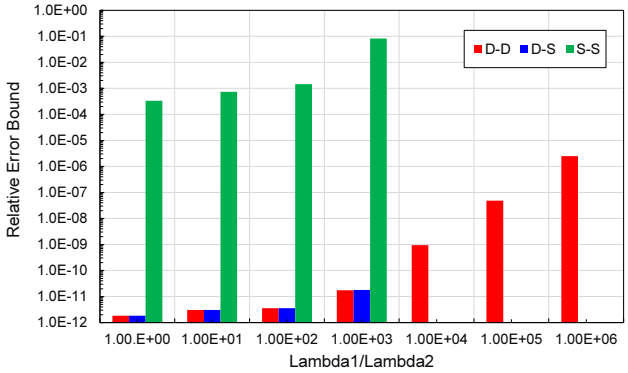

図 31　OBCX で求められた相対最大誤差

7. まとめ

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には，メモリ・ネットワークの階層の深化に対応した通信最適化（Serial，Parallel）とともに省電力・省エネルギーに向けた検討が必要である．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，計算時間を短縮し省電力・省エネルギーを図る試みが活発に行われている．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が，実問題で現れる大規模疎行列･H 行列への応用例はほとんどない．本研究では，有限体積法から導出される大規模疎行列を係数行列とする連立一次方程式の求解に着目し，疎行列格納手法，問題規模と低精度演算による性能改善の関係に注目し，様々な計算機環境下での検討を実施した．著者等によって提案された，M 行列性を有する疎行列を係数行列とする連立一次方程式向けの実用的な精度保証手法〔10,11〕を適用し，精度保証を実施した．

本研究では，東京大学情報基盤センターに導入された Oakleaf-7（Fujitsu PRIMEHPC FX700，A64FX を 8 基搭載）を使用して，性能評価，特に半精度演算を含む低精度，混合精度演算の評価を実施した．本文中にも述べたように，現状では「`!$omp simd`」の動作に問題があり，充分な性能が得られていないものの，前処理付き反復法の前処理部に半精度演算を導入することの有効性が示された．

本研究では，ほとんどの計算を疎行列格納法として CRS 法を使用したプログラムによって実施したが，先行研究〔7〕でも明かとなったように，低精度演算導入による計算速度向上の効果は演算密度が高い場合に顕著である．今回は各手法の有効性を示すための予備的な研究であったが，今後はより複雑であるが高い性能を実現できる ELL，ELLo，SCS，OPT 等の実装を適用した評価を実施していく必要がある．実際，Oakleaf-7（FX700）では 1 反復当たりの計算時間は，倍精度⇒単精度によって，演算密度の高い OPT 形式では 7%の短縮が見られているのに対して，演算密度の低い CRS では逆に 10%以上遅くなっている．

本文中でも述べたように，半精度演算（FP16）は有効桁数が 3 桁程度のため，実問題に使用することは困難である．山口等はは半精度と単精度（FP32）の中間的な特性を持つデータ型である FP21 を提案し，有限要素解析における前処理において活用している〔25〕．FP21 は FP32 と同じレンジを持つもののデータサイズが 2/3 となるため，メモリバンド幅ネックのカーネルにおいては実行時間が 2/3 になると期待される．現在の主要な計算機においては FP21 の演算器はサポートされていないため，〔25〕では変数のメモリへの格納時のみに FP21 を使い，計算時には FP21 を FP32 に変換して演算している．これらの成果は GPU 上で実装され，OpenACC 版が公開されている〔26〕．FP21 等の適用も含めて混合精度演算活用を検討していく必要がある．

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．AMD Rome 環境を貸与いただいた中村正澄氏他日本 AMD の皆様には深く感謝いたします．また，Oakleaf-7 利用に当たってアドバイスを頂いた中澤淳氏，坂口吉生氏（富士通）に深く感謝いたします．

参考文献

1) 中島研吾，拡張型Sliced-ELL行列格納手法に基づくメニィコア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) 中島研吾，荻田武史，パイプライン型アルゴリズムによる並列共役勾配法の安定性評価，情報処理学会研究報告（2018-HPC-167-26）， 2018
6) 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
7) 中島研吾，坂本龍一，星野哲也，有間英志，塙敏博，近藤正章，低精度演算とアプリケーション性能，情報処理学会研究報告（2020-HPC-174-5），2020
8) Mittal, A., A Survey of Techniques for Approximate Computing, ACM CSUR 48-4, 2016
9) 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
10) Ogita, T., Nakajima, K., Accurate and verified solutions of large sparse linear systems arising from 3D Poisson equation, International Conference on Matrix Analysis and its Applications (MAT TRIAD 2019), Liblice, Czech Republic, September 10, 2019)
11) Ogita, T., Nakajima, K., Verified solutions of large sparse linear systems arising from 3D Poisson equation in HPC environments, European Numerical Mathematics and Advanced Applications Conference 2019 (EnuMath 2019) (Egmond aan Zee, The Netherlands, October 2, 2019)
12) 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
13) 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
14) Nakajima, K., Optimization of Serial and Parallel Communications for Parallel Geometric Multigrid Method, IEEE Proceedings of 20th IEEE International Conference on Parallel and Distributed System (ICPADS 2014), 25-32, 2014
15) 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
16) 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
17) 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/
18) Ogita, T., Oishi, S., Ushiro, Y., Fast Verification of Solutions for Sparse Monotone Matrix Equations, Computing 15, 175-187, 2001
19) Ogita, T., Oishi, S., Ushiro, Y., Fast Inclusio and Residual Iteration for Soutions of Matrix Equations, Computing 16, 171-184, 2002
20) 最先端共同 HPC 基盤施設，http://jcahpc.jp/
21) AMD，https://www.amd.com/
22) 富士通株式会，https://www.fujitsu,com
23) STREAM Benchmark: https://www.cs.virginia.edu/stream/
24) 細川洋輝，奥田洋司，階層的メモリ構造を考慮した反復法ソルバーの混合精度計算，第 24 回日本計算工学会講演会（大宮，2019 年 5 月 29 日），2019
25) Yamaguchi, T., Fujita, K., Ichimura, T., Naruse, A., Lalith, M., Hori, M., GPU implementation of a sophisticated implicit low-order finite element solver with FP21-32-64 computation using OpenACC, Proceedings of Sixth Workshop on Accelerator Programming Using Directives (WACCPD) in conjunction with SC19 (The International Conference for High Performance Computing, Networking, Storage, and Analysis), 2019
26) FP21AXPY OpenACC source code, https://github.com/y-mag-chi/fp21axpy