Vol.2022-HPC-185 No.26
2022/7/29

多相 CFD シミュレーションにおける悪条件行列に対する GPU 向け BFloat16 精度前処理の検証

伊奈拓也[1] 井戸村泰宏[1] 今村俊幸[2] 山下晋[1] 小野寺直幸[1]

概要：「富岳」や「Summit」をはじめとする最先端スーパーコンピュータでは倍精度演算性能よりも低精度演算性能の方が数倍高く，FP16 や Bfloat16 を活用した混合精度処理が有効である．しかし，多相 CFD シミュレーションの悪条件行列に対する反復解法に低精度演算をそのまま適用すると収束性の悪化を引き起こす問題がある．これまで，多相熱流動解析コード JUPITER を対象としてクリロフ部分空間法の混合精度前処理を構築し，A64FX では行列のスケーリングと FP16 データ/FP32 演算混合精度前処理により収束性の維持と高速化を実現した．本研究では、NVIDIA GPU でサポートされている BFloat16 を用いて混合精度前処理の検証を行った．Bfloat16 は FP32 と同等なダイナミックレンジを持つため FP16 では必須であるオーバーフローを防ぐためのスケーリングは不要である．その結果、Bfloat16 を用いることで FP16 を使用した場合と比較して前処理で 7%の高速化を確認した．しかし，仮数部のビット数が FP16 よりも少ない Bfloat16 では収束性が悪化するケースも見られた．

キーワード：クリロフ部分空間法，低精度演算

1. はじめに

低精度演算を利用した科学技術計算の高速化が注目されている．機械学習アプリケーションでは倍精度演算性能よりも半精度演算性能の需要が高いため，「富岳」，「Summit」及び「Frontier」など最先端のスーパーコンピュータでは倍精度演算性能よりも低精度演算性能が重視されている．そのため，科学技術計算においても FP16 や bfloat16(BF16)を利用した低精度演算を利用することが重要となっている．

このような背景から，近年，低精度演算を活用した行列計算の高速化手法が研究されてきた．[1][2][3]では FP16 を用いた LU 分解の高速化と反復改良法(IR 法)による精度向上を組み合わせた混合精度演算により，倍精度と同等な精度と高速化が密行列の線形ソルバで実現された．[4]ではブロックヤコビ法(BJ 法)においてブロックの条件数に応じて FP16，FP32，FP64 を適宜切り替える手法が提案された．この手法は SuiteSparse matrix collection [5]の行列に対してテストされ，メモリアクセスおよび消費電力の削減が示された．[6]では符号部 1 ビット，指数部 8 ビット，仮数部 12 ビットで構成される FP21 形式が提案された．有限要素法の疎行列線形ソルバの共役勾配法(CG 法)のベクトルを FP21 で保存することで数値精度の維持と高速化を実現した．[7]では核融合プラズマ乱流計算の線形ソルバである CA-GMRES 法[8]の対称逐次緩和法前処理[9]に FP16 を適用し，混合精度演算による高速化を実現した．[10]では多相 CFD シミュレーションの悪条件行列を解く線形ソルバである CG 法の前処理に FP16 を適用した．悪条件行列における丸め誤差の影響による収束性悪化を回避するため，FP16 データと FP32 演算を組み合わせた FP16 データ/FP32 演算混合精度前処理を開発した．FP16 データを FP32 に変換して計算を行うことで丸め誤差を低減させて高速化と収束性の維持を達成した．また，[10]では収束性を悪化させないために必要な仮数部ビット数として 7 ビットあれば十分であることを検証しており，混合精度前処理が BF16 に対しても有効な手法であることを確認している．

本研究では，多相 CFD シミュレーションにおける悪条件行列に対して BF16 を用いた混合精度演算を適用して BF16 の有用性を検証する．

2. 多相多成分熱熱流動解析コード JUPITER

日本原子力研究開発機構では原子炉における過酷事故時の炉内溶融物の移行挙動を再現することを目的にした多相多成分熱流動解析コード JUPITER[11]の開発を進めている．JUPITER では，原子炉内の核燃料，構造物，気体などを含む複数成分の固気液 3 相の熱流動を取り扱う．そのため，大きなコントラストの密度を係数に含む圧力ポアソン方程式の行列は条件数が 10^9 を超える悪条件問題となり，圧力ポアソン方程式を解く計算コストが 90%以上を占める．圧力ポアソン方程式は 3 次元直交格子(x，y，z)上で 2 次精度の中心差分法により離散化されている．

$$\nabla \cdot \left(\frac{1}{\rho}\nabla p\right) = \frac{1}{\Delta t}\nabla \cdot u \qquad (1)$$

ここで、ρは密度，pは圧力，Δtは時間ステップ，uは速度を表す．

ポアソンソルバとして BJ 前処理付 CG 法を採用している．3 次元領域分割法を用いて並列化されており CPU 版では MPI+OpenMP，GPU 版では MPI+CUDA によりハイブリッド並列化されている．

1 日本原子力研究開発機構 システム計算科学センター
2 理化学研究所 計算科学研究センター

Vol.2022-HPC-185 No.26
2022/7/29

3. ポアソンソルバ

3.1 前処理付き共役勾配法

前処理付き共役勾配法(PCG 法)のアルゴリズムを図 1 に示す．1，7 行目の M^{-1}の計算が前処理において，BJ 法で分割した領域ごとに不完全 LU 分解(ILU)を計算する．

1 $r_1 := b - Ax_1$，$z_1 := M^{-1}r_1$，$p_1 := z_1$
2 $for\ j = 1,\ 2,\ \ldots,\ until\ converge$
3 $\omega := Ap_j$
4 $\alpha_j := \langle r_j,\ z_j\rangle / \langle \omega,\ p_j\rangle$
5 $x_{j+1} := x_j + \alpha_j p_j$
6 $r_{j+1} := r_j - \alpha_j p_j$
7 $z_{j+1} := M^{-1} r_{j+1}$
8 $\beta_j := \langle r_{j+1},\ z_{j+1}\rangle / \langle r_j,\ z_j\rangle$
9 $p_{j+1} := z_{j+1} + \beta_j p_j$
10 $end\ for$

図 1 前処理付き共役勾配法(PCG 法)

3.2 GPU 向け前処理用データ構造

オリジナルの CPU 版の JUPITER は図 2 の z-division により z 方向に分割して 1 ブロックに 1 スレッド割り当ている．各ブロックを ILU で計算するため，ILU のデータ依存性によりベクトル化が難しく，SIMD 演算が適用されていない．

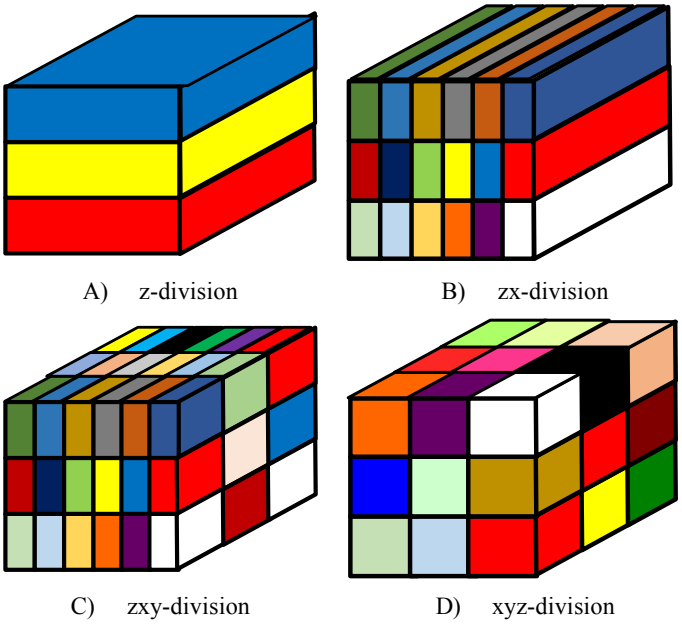
富岳向けに最適化された A64FX 版の JUPITER[10]は図 2 の zy-division により分割されている．BJ 法で x 方向のブロックサイズを 1 に固定して x 方向のデータ依存性を解消させることでソフトウェアパイプライニングと SIMD 演算を適用させている．z 方向の分割は z-division と同様に OpenMP により並列化している．

GPU 版の JUPITER[12]は図 2 の zxy-division により分割している． A64FX 版と同様に x 方向のブロックサイズを 1 に固定することでデータ依存性のない連続メモリアクセスを実現している．さらに，z 方向に加えて y 方向も分割することでブロック数を増やし，1 ブロックに 1CUDA スレッドを割り当てることで効率的な GPU 計算を実現している．

zy-division，zxy-division は x 方向のデータ依存性を無視することで各アーキテクチャで効率的に計算を行う．しかし， 物理的な特徴を考慮せずに x 方向の影響を無視するため計算モデルにより収束性が低下する可能性がある．加えて，GPU 版では A64FX 版よりもブロックサイズが小さくなることで収束性悪化の問題がより顕著になる可能性がある．

そこで，GPU 向けに BJ 法のブロックを 3 次元で分割する xyz-division を採用する．3 次元でブロックを分割することで特定の方向の影響を無視する zxy-division よりも計算モデルに起因する前処理への影響は少なくなる．一方，3 次元ブロック分割ではブロック数が少なくなり，各ブロックへのメモリアクセスがストライドアクセスになるという課題がある．この課題を解決するために，本研究では GPU の BJ 前処理向けにデータ構造を変更した． 図 3 に構造格子のオーダリング例を示す．JUPITER の BJ 法は太枠で囲まれたブロックを ILU で解く． xyz-division ではメモリアクセスが連続になる x 方向にデータ依存性があるため GPU で効率的に計算することが困難となる．

データ依存性を緩和するために格子番号の並び替えを行う．ブロックごとにサイクリックに番号を振り直すことで，異なるブロックの同じ位置の格子点を同時に計算可能となる．また，メモリに連続アクセスすることが可能となり GPU で効率的に計算することが可能となる．



A) z-division　B) zx-division
C) zxy-division　D) xyz-division

図 2 ブロックヤコビ法のブロック形状

73	74	75	76	77	78	79	80	81
64	65	66	67	68	69	70	71	72
55	56	57	58	59	60	61	62	63
46	47	48	49	50	51	52	53	54
37	38	39	40	41	42	43	44	45
28	29	30	31	32	33	34	35	36
19	20	21	22	23	24	25	26	27
10	11	12	13	14	15	16	17	18
1	2	3	4	5	6	7	8	9

A) 1 次元オーダリング

61	70	79	62	71	80	63	72	81
34	43	52	35	44	53	36	45	54
7	16	25	8	17	26	9	18	27
58	67	76	59	68	77	60	69	78
31	40	49	32	41	50	33	42	51
4	13	22	5	14	23	6	15	24
55	64	73	56	65	74	57	66	75
28	37	46	29	38	47	30	39	48
1	10	19	2	11	20	3	12	21

B) ストライドオーダリング

図 3 構造格子のオーダリング例

3.3 反復改良前処理

A64FX 版では富岳向けにブロックサイズを変更したことで収束性が悪化した．この問題を解決するために反復改良法(IR 法)を組み合わせた IR 前処理を行うことで収束性を改善させた．図 4 に IR 前処理のアルゴリズムを示す．クリロフ部分空可能法の前処理では行列Aの近似行列であるMにより$Mz = r$を解くことで行列Aの近似解を計算する．JUPITER の IR 前処理では ILU による直接法で得た近似解を IR 法で修正することで収束性を向上させている．GPU 版でも IR 前処理を採用することで収束性を向上させる．反復法であるため，任意の要求精度になるまで反復計算をすることが可能であるが JUPITER では 1 回の反復で打ち切る．

1 $Mz = r, M \sim A$
2 $z_1 := M^{-1} r$
3 $for\ j = 1,2, \dots, n$
4 $\quad u_j := r - A z_j$
5 $\quad y_j := M^{-1} u_j$
6 $\quad z_j := z_j + y_j$
7 $end\ for$

図 4 反復改良前処理

3.4 混合精度前処理

GPU 版 JUPITER におけるオリジナルの PCG 法は FP64 で計算している．本研究では，低精度演算を活用するために前処理を低精度で行う混合精度前処理を実装する．

3.4.1 FP16

FP16 は符号部 1 ビット，指数部 5 ビット，仮数部 10 ビットで構成される．FP32 よりも指数部のビット数が少ないためダイナミックレンジが狭くオーバーフロー，アンダーフローが起こりやすい．これを防ぐために A64FX 版では，前処理行列Mの各行の最大値ノルムを対角要素に持つ対角行列Dにより$\tilde{M} := D^{-1}M$の最大値を 1 にするスケーリングを行っている．これにより行列のオーバーフローを防ぐ．富岳版のスケーリング方法を図 5 に示す．

図 6 に示す GPU 版ではスケーリング後の行列についても対称性を保つように$\tilde{M} := D^{-1/2}\, MD^{-1/2}$でスケーリングする．対称スケーリングにすることで残差計算及びILU計算で使用する係数行列で対称性を利用することができるようになるためメモリアクセスが削減される．

ベクトルに対しては BJ 法の各ブロックの最大値により$\tilde{r} := D^{-1/2}\, r / \left\| D^{-1/2} r \right\|$でスケーリングする．スケーリングされるベクトル$r$は残差ベクトルであるため反復の初期はオーバーフローを防ぐためにスケーリングを行う．一方，反復が進み残差ベクトルが小さくなると低精度に変換した際にアンダーフローが発生するため，スケーリングを行い情報が失われるのを防ぐ．

A64FX 版では集団通信を回避するためにプロセスごとにスケーリング係数$\|D^{-1}r\|$を変えていた. GPU 版ではブロックごとにスケーリング係数$\left\|D^{-1/2}r\right\|$を変えることでスレッドブロック間の同期を回避する．また，極端に大きなスケーリング係数によりスケーリング後の値がゼロとなり情報を失う影響をブロック内に限定することで収束性を安定させる．

最後にスケーリングした線形方程式$\tilde{M}\tilde{y} := \tilde{r}$の解$\tilde{y}$を$z := D^{-1/2}\left\|D^{-1/2}r\right\|\tilde{y}$でスケーリングを戻すことでオーバーフロー，アンダーフローを回避して前処理を行う．

A64FX 版では FP16 による丸め誤差を低減させるために，FP16 データを FP32 に変換して中間結果を FP32 で保存して最終結果を FP16 に変換する FP16 データ/FP32 演算混合精度前処理[10]により前処理を行っている．GPU 版も丸め誤差低減のために FP16 データ/FP32 演算混合精度前処理を採用する．

1 $Mz = r, M \sim A$
2 $D_{ii} := \max(|M_{i1}|, |M_{i2}|, \dots, |M_{in}|)$
3 $\tilde{r} := D^{-1}\, r / \left\| D^{-1} r \right\|$
4 $\tilde{M} := D^{-1} M$
5 $\tilde{y}_1 := \tilde{M}^{-1} \tilde{r}$
6 $for\ j = 1,2, \dots, n$
7 $\quad \tilde{u}_j := \tilde{r} - D^{-1} A \tilde{y}_j$
8 $\quad \tilde{v}_j := \tilde{M}^{-1} D^{-1}\, \tilde{u}_j / \left\| D^{-1} \tilde{u}_j \right\|$
9 $\quad \tilde{y}_j := \tilde{y}_j + \left\| D^{-1} \tilde{u}_j \right\| \tilde{v}_j$
10 $end\ for$
11 $z := \left\| D^{-1} r \right\| \tilde{y}_n$

図 5 スケーリング付き反復改良前処理

1 $Mz = r, M \sim A$
2 $D_{ii} := \max(|M_{i1}|, |M_{i2}|, \dots, |M_{in}|)$
3 $\tilde{r} := D^{-1/2}\, r / \left\| D^{-1/2} r \right\|$
4 $\tilde{M} := D^{-1/2} M D^{-1/2}$
5 $\tilde{y}_1 := \tilde{M}^{-1} \tilde{r}$

6 $for\ j = 1,2, \dots, n$

7 $\tilde{u}_j := D^{-1/2}\tilde{r} - D^{-1/2}AD^{-1/2}\tilde{y}_j$

8 $\tilde{v}_j := \tilde{M}^{-1}D^{-1/2}\,\tilde{u}_j / \left\|D^{-1/2}\tilde{u}_j\right\|$

9 $\tilde{y}_j := \tilde{y}_j + \left\|D^{-1/2}\tilde{u}_j\right\|\tilde{v}_j$

10 $end\ for$

11 $z := D^{-1/2}\left\|D^{-1/2}r\right\|\tilde{y}_n$

図 6 対称スケーリング付き反復改良前処理

3.4.2 BFloat16

BF16 は符号部 1 ビット，指数部 8 ビット，仮数部 7 ビットで構成される. FP32 と同じ指数部ビットを持つためダイナミックレンジは FP32 と同等である．そのため，FP16 で必要であったスケーリングを行う必要がない．しかし，仮数部が FP16 よりも少ないため丸め誤差による影響を受けやすい.

前処理は FP16 と同様に BF16 を FP32 に変換して中間結果を FP32 で計算して最終結果を BF16 に変換する BF16 データ/FP32 演算混合精度前処理により行う.

4. 性能測定

性能測定は日本原子力研究開発機構が所有する SGI8600 GPGPU 演算部において実施した. SGI8600 の諸元を表 1 に示す.

本研究ではバンドル体系における気液二相流の解析を対象として性能評価を実施した．計算モデル全系を図 7，格子サイズごとの xy 平面を図 8 に示す．赤い領域が燃料棒であり，4×4 本の燃料棒を模擬したバンドル体型となっている.

表 1 SGI8600 GPGPU 演算部諸元(1 ノード)

CPU	プロセッサ名	Intel Xeon Gold 6248R
	プロセッサ数 (コア数)	2 (24+24)
	演算性能	4.6[TFLOPS/node]
	メモリバンド幅	281.6 [GB/s/node]
GPU	プロセッサ名	NVIDIA Tesla V100 SXM2
	プロセッサ数	4
	演算性能	31.2[TFLOPS/node]
	メモリバンド幅	3.6[GB/s/node]
	GPU 間接続	NVLink(25GB/s)
ノード間接続		Infiniband 4xEDR x 4 本 (片方向 12.5GB/s/本)

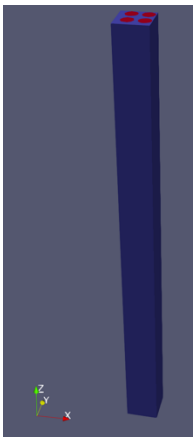
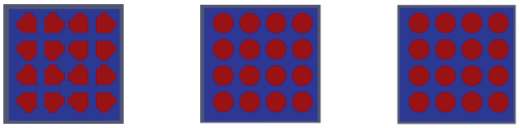


図 7 計算モデル全系



28×28×750 格子　112×112×3000 格子　224×224×6000 格子

図 8 計算モデル(xy 平面)

4.1 ルーフライン性能

混合精度前処理の性能をルーフラインモデル[13]により評価した．評価結果を表 2 に示す．格子サイズは 112×112×3000 ，BJ 法のブロックサイズは 4×4×5 として SGI8600 を 1 ノード使用して計測した．ルーフライン性能はルーフラインモデルで評価した 1 反復あたりの理論計算時間．計算時間は測定した PCG 法 1 反復あたりの計算時間である． 計算時間をルーフライン性能と比較すると axpy は 69%, SpMV は 48%, 前処理は FP32 で 32%, FP16, BF16 共に 33%となった．データ構造を変更して前処理のデータ依存性を緩和したが，SpMV と前処理は理論性能の 30%程度の低い性能であり，性能改善の余地がある.

表 2 ルーフライン性能

	演算数 [flop]	メモリアクセス [byte]	ルーフライン性能 [ms/iter]	計算時間 [ms/iter]
axpy	4	40	0.42	0.61
SpMV	15	56	0.60	1.24
preconditioning (FP16)	59	90	1.01	3.08
preconditioning (BF16)	50	86	0.96	2.85
preconditioning (FP32)	50	116	1.27	4.01

4.2 収束性評価

4.2.1 ブロックサイズによる評価

表 3 に BJ 法のブロックサイズを変更して収束性を評価した結果を示す．格子サイズは 224×224×6000 である.

z-division では 3 次元にブロックを分割するため時間ステ

ップが進んでも収束性が維持されている． zxy-division では x 方向のブロックが 1 に固定されるため 2 ステップ目で収束しなくなっている．この傾向はブロックサイズを大きくしても変わらない．一方，xyz-division は 3 次元に分割したことに加え IR 前処理を採用したことで z-division よりも収束性が向上している．

表 3 ブロックサイズによる収束性の比較

時間ステップ	CPU 版 z-division 112×112×75	GPU 版 zxy-division 1×14×15	GPU 版 zxy-division 1×112×375	GPU 版 xyz-division 4×4×5 FP32 IR 前処理
1	23097	37705	35976	16997
2	38745	未収束	未収束	21330
3	20007	*	*	15153

4.2.2 前処理精度

表 4 に各格子サイズの収束までの反復回数を示す．全ケースで BJ 法のブロックサイズは 4×4×5 である.格子サイズ 112×112×3000, 224×224×6000 では全ての前処理で同程度の反復回数で収束している．しかし，28×28×750 格子では BF16 前処理のみ収束していない．

表 4 前処理精度による反復回数の比較

	FP32 前処理	FP 16 前処理	BF16 前処理
28×28×750 格子 反復回数	1641	1650	未収束
112×112×3000 格子 反復回数	7651	7773	7846
224×224×6000 格子 反復回数	16997	16998	17086

図 9 に前処理に利用する行列データを FP64 から低精度に変換する際の丸めモードを変更した場合の収束履歴を示す． 格子サイズは 28×28×750 である．最近接丸めは最も近い値に丸められる．ゼロ方向丸めはゼロに近い方向に値を丸める方法であり絶対値が元の値より小さくなる．

FP16 前処理は最近接丸め, ゼロ方向丸めの両方で収束している．一方、BF16 前処理はゼロ方向丸めでは収束しているが最近接丸めでは途中で収束が停滞している．仮数部の精度が 3 桁程度の FP16 に対して、仮数部の精度が 2 桁程度まで低下する BF16 では丸め誤差の影響が FP16 よりも大きいため丸め方法により収束性が変化している．また，ゼロ方向丸めでは BF16 前処理は収束しているが FP16 前処理よりも収束性が悪化している．

ポアソン方程式の 7 重ブロック対角行列の係数は密度，時間ステップ幅，および離散化スキームによって決まる．ゼロ方向丸めの場合には，全ての係数の仮数部が-方向に丸められて平均的に絶対値が小さな行列に変換されるが行列係数のコントラストの構造は維持されているものと考えられる．一方，最近接丸めの場合には，各係数の仮数部に応じて+方向と-方向の丸め処理がランダムに発生し，行列係数のコントラストの構造が変化し，これが条件数に悪影響を与えているものと考えられる．

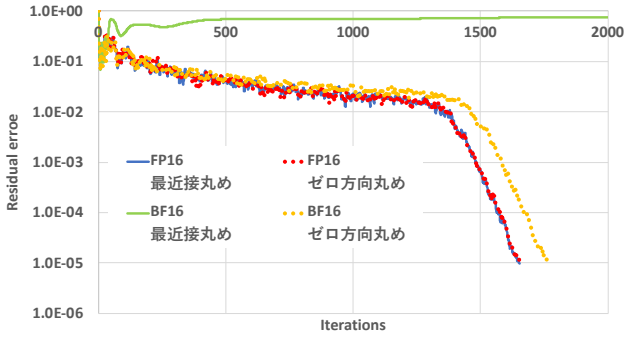


図 9 FP64 行列データを低精度に変換する際の丸める方法による収束履歴の比較

4.3 前処理性能比較

前処理性能を比較した結果を表 5 に示す． 格子サイズは 224×224×6000 ， BJ 法のブロックサイズは 4×4×5，SGI8600 を 2 ノード使用して計測した．BF16 前処理では反復回数が FP16 よりも増加しているが計算時間は FP16 前処理よりも 7%高速である. BF16 ではスケーリングが不要であるため FP16 よりも高速となっている．

表 5 前処理性能比較

	FP32 前処理	FP16 前処理	BF16 前処理
前処理 [s]	275	217	202
全計算時間 [s]	450	392	378
反復回数	16997	16998	17088

5. おわりに

本研究では多相多成分熱流動解析コード JUPITER に BF16 を利用した混合精度前処理を適用した．前処理に BF16 を利用しても FP32 と同等な収束性を保つことが可能であることを確認した．また，指数部ビット数が FP32 と同じ BF16 を利用することでスケーリングが不要となり FP16 よりも高速となることを確認した．しかし，仮数部 7 ビットの BF16 では丸め誤差による影響が強く丸め方法によって収束しないケースが存在した．

今後の課題として PCG 法の前処理以外の処理に対して低精度演算の適用を考える．また，FP64 行列データを低精度に変換する丸め方法について検証したが，(FP16，BF16) データ/FP32 演算混合精度前処理の丸め方法についても最近接丸め以外の Stochastic rounding[14]などの丸め方法の検証を行いよりロバストな混合精度前処理の開発を目指す．

謝辞 本研究は科学研究費補助金・基盤研究(C) (課題番号 22K12053)の支援を受けました. この研究では JAEA が提供する SGI8600 の計算リソースを使用しました．

参考文献

[1] E. Carson, N. J. Higham,“A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems,”SIAM Journal on Scientific Computing,2017,vol. 39,no. 6,pp. A2834‐A2856.
[2] A. Haidar,S. Tomov,J. Dongarra et al.,“Harnessing gpu tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers,”in SC18: International Conference for High Performance Computing,Networking,Storage and Analysis, 2018,pp. 603‐613.
[3] S. Kudo,K. Nitadori,T. Ina et al., “Implementation and numerical techniques for one eflop/s hpl-ai benchmark on fugaku,” in 2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA),2020,pp. 69‐76.
[4] H. Anzt,J. Dongarra,G. Flegar et al., “Adaptive precision in block-jacobi preconditioning for iterative sparse linear system solvers,” Concurrency and Computation: Practice and Experience,2019,vol. 31,no. 6,p. e4460, e4460 cpe.4460.
[5] T. A. Davis and Y. Hu, “The university of florida sparse matrix collection,” ACM Trans. Math. Softw.,Dec. 2011,vol. 38,no. 1.
[6] T. Ichimura,K. Fujita,T. Yamaguchi et al., “A fast scalable implicit solver for nonlinear time-evolution earthquake city problem on lowordered unstructured finite elements with artificial intelligence and transprecision computing,” in Proceedings of the International Conference for High Performance Computing,Networking,Storage,and Analysis,ser. SC ’18. IEEE Press,2018.
[7] Y. Idomura,T. Ina,Y. Ali et al., “Acceleration of fusion plasma turbulence simulations using the mixed-precision communication-avoiding krylov method,” in SC20: International Conference for High Performance Computing,Networking,Storage and Analysis,2020,pp. 1‐13.
[8] E. C. Carson, “Communication-avoiding krylov subspace methods in theory and practice,” Ph.D. dissertation,University of California,Berkeley,2015.
[9] Y. Saad,Iterative Methods for Sparse Linear Systems,2nd ed. Philadelphia,PA,USA: Society for Industrial and Applied Mathematics,2003.
[10] Takuya Ina,Yasuhiro Idomura,Toshiyuki Imamura,Susumu Yamashita,Naoyuki Onodera: Iterative methods with mixed-precision preconditioning for ill-conditioned linear systems in multiphase CFD simulations. in 2021 IEEE/ACM 12th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA21), 2021,p1-8
[11] S. Yamashita,T. Ina,Y. Idomura,and H. Yoshida, “A numerical simulation method for molten material behavior in nuclear reactors,” Nuclear Engineering and Design,2017,vol. 322,pp. 301 ‐ 312.
[12] Y. Ali,N. Onodera,Y. Idomura et al., “Gpu acceleration of communication avoiding chebyshev basis conjugate gradient solver for multiphase cfd simulations,” in 2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA),2019,pp. 1‐8.
[13] T. Shimokawabe,T. Aoki,C. Muroi et al., “An 80-fold speedup,15.0 tflops full gpu acceleration of non-hydrostatic weather model asuca production code,” in SC ’10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing,Networking,Storage and Analysis,2010,pp. 1‐11.
[14] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan,” Deep Learning with Limited Numerical Precision”,Proceedings of the 32nd International Conference on Machine Learning, 2015, PMLR 37:1737-1746.