

低精度浮動小数点数を適用したICCG法の性能評価

河合 直聡^{1,a)} 中島 研吾^{1,2}

概要: 本研究では ICCG 法の IC 前処理に低精度演算を適用し、計算時間短縮の効果および収束性への影響を Wisteria-BDEC および Oakforest-PACS、Oakbridge-CX 上で評価した。ICCG 法などの反復法では、高い演算精度が必要とされるため、倍精度浮動小数点数の利用が一般的である。ただし、ICCG 法などの反復法はメモリ律速なアプリケーションであるため、より低精度な浮動小数点数が適用できれば、計算時間短縮の効果が期待できる。本研究では IC 前処理部の行列およびベクトルを格納する浮動小数点数を低精度化し、計算時間短縮の効果および収束性への影響を評価した。

1. はじめに

近年の科学技術分野では倍精度浮動小数点数 (FP64) 以外に単精度浮動小数点数 (FP32) や半精度浮動小数点数 (FP16) などの低精度浮動小数点数の利用が再検討され、計算時間短縮、消費電力削減を目的として複数の精度を混在させる研究が行われている。現在の数値シミュレーションなど分野では暗黙的に FP64 が利用されているが、深層学習の分野を中心に FP32 や FP16 の使用が広く議論、実用化されている [1], [2]。実際に、近年の GPU では半精度演算がサポートされ、また 2020 年春から試験稼働を開始している富岳で採用されている CPU (ARM64Fx) でも同様にサポートされている。加えて、第 3 世代の Intel Xeon Scalable Processor でも FP16 のサポートがアナウンスされている。このような背景を受け、計算機シミュレーションの分野でも FP32 の利用が再検討されており、FP32 がアプリケーションによっては実用的な精度であると報告されているが [3], [4], [5]、FP16 のほとんどない。

数値シミュレーション内の陰解法のソルバとして広く用いられているクリロフ部分空間法は、FP64 の利用が一般的とされている典型的な例である。これは、クリロフ部分空間法の収束性が計算精度に強く影響を受けるためである。しかし、クリロフ部分空間法は一般的にメモリバンド幅律速なアプリケーションであるため、低精度演算の利用により演算時間短縮の効果が期待できる。特に、クリロフ部分空間法で併用される前処理部分に着目すると、ここでは簡

易的に対象の方程式が解ければよいという性質から、低精度でもある程度の効果が期待できる。不完全コレスキー分解 (IC) 前処理は、反復法全体の計算時間の半分以上を締め、またヒューリスティックな前処理であるために、低い精度でも十分な収束性改善が確認されており、実際に計算時間の短縮に寄与している [6]。

本研究では pHEAT-3D アプリケーション [7] の IC 前処理に FP16 を含む低精度浮動小数点数を適用し、計算時間短縮および収束性への影響を評価した。本評価は前処理で参照する行列とベクトルの浮動小数点数をそれぞれ低精度化した。低精度化した状態でもメモリへの十分なアクセスを確保するため、本アプリケーションでの疎行列の格納形式は CRS だけでなく、ELL [8] や Sell-C- σ [9] でも実装し、評価対象とした。また、収束性への影響は、pGW3D-FVM から導出される問題の係数行列に関して、条件数が小さい良条件から条件数の大きな悪条件までの行列セットを想定し、評価を行った。評価では Oakforest-PACS、Oakbridge-CX、Wisteria/BDEC01 odyssey スーパーコンピュータの 1 ノードを使用した。

2. 半精度浮動小数点数

図 1 に FP64、32、16 のビット表現上の差を、表 1 には各精度の表現能力を示す指標として、仮数部の 10 進数での表現可能桁数と指数部の最大冪指数を示す。なお、本表での仮数部の 10 進数での表現可能桁数 y は、各精度の仮数部のビット数を x として、

$$10^y = 2^{(x+1)} \quad (1)$$

$$y = (x+1) \log_{10} 2 \quad (2)$$

のように算出している。なお、式 (1) の右辺、2 進数での

¹ 東京大学 情報基盤センター
Information Technology Center, Kashiwa 277-0882, Japan

² 理化学研究所 計算科学研究センター
RIKEN R-CCS

^{a)} kawai@cc.u-tokyo.ac.jp

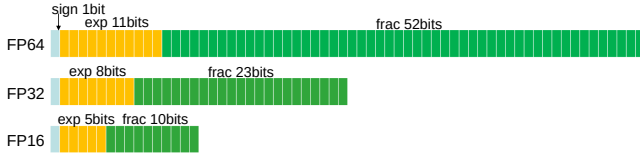


図 1 FP64,32,16 の比較

表 1 各浮動小数表現の表現力

浮動小数点数	仮数部：表現可能桁数	指数部：最大冪指数
FP16	3.31	5
FP32	7.22	38
FP64	15.95	308

表現可能桁数を示す冪指数を $x+1$ としているのは、浮動小数表現では、仮数部の最上位ビットのさらに上に 1 が常に付与されているとして (Hidden Bit)、浮動小数形式を扱うためである。

FP16 は FP32 に対して半分のビット数で浮動小数を表現しており、仮数部の表現可能桁数は 3 桁程度と FP32 の半分程度、最大冪指数は 5 と 1/7 程度になっている。反復法では特に冪指数小ささが影響すると考えられるため、注意して使用する必要がある。

3. pHEAT-3D アプリケーション

三次元定常熱伝導解析シミュレーション (pHEAT-3D) を有限体積法 (FVM) を用いて離散化し、導出される連立一次方程式を ICCG の法を用いて解くことを考える。ICCG 法での低精度浮動小数点数による収束性への影響と計算時間短縮の効果を評価するためには、解くべき問題の係数行列に関して、良条件から、悪条件まで行列セットを想定する必要がある。pHEAT-3D では温度拡散率を空間的に不均一とすれば複数の条件の行列セットを生成可能である。本説では pHEAT-3D で係数行列の導出と条件の異なる行列セットの作成方法、および ICCG 法への低精度浮動小数点数の適用方法について述べる。

3.1 pHEAT-3D での係数行列の性質

pHEAT-3D では、一般的なポアソン方程式である次の式を、有限体積法 (FVM) を用いて離散化している。

$$\nabla \cdot (\lambda \nabla \phi) = \rho, \quad \phi = 0 \text{ at } z = z_{max} \quad (3)$$

ここで、 ϕ は温度分布を、 ρ は発熱量を示す。また、 λ は温度拡散率を示し、空間的に不均一な値となっている。温度拡散率は解析対象となる物質の密度、比熱、熱伝導率から導出され、その最大値最小値は明確ではないが、十分に実用的な範囲を内包するために、本問題では $\lambda = 10^{-5} \sim 10^5$ とする。これを格子数が $nx \times ny \times nz$ の構造格子上で、FVM を用いて離散化した場合、 ϕ および ρ の離散空間上のベクトルを $\mathbf{x}$ 、 $\mathbf{b}$ として、解くべき方程式 $\mathbf{Ax} = \mathbf{b}$ の係数行列 $\mathbf{A}$ の要素 $a_{i,j}$ は次のように導出される。

do $k = 1$, !until converge

$$\alpha = \frac{(\mathbf{r}^k, \mathbf{p}^k)}{(\mathbf{p}^k, \mathbf{Ap}^k)}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha \mathbf{p}^k$$

$$\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha \mathbf{Ap}^k$$

$$\mathbf{r} = \mathbf{r}^{k+1}$$

$$\mathbf{q} = \mathbf{U}^{-1} \mathbf{D}^{-1} \mathbf{L}^{-1} \mathbf{r} \quad \text{!不完全コレスキー分解前処理}$$

$$\beta = -\frac{(\mathbf{q}, \mathbf{Ap}^k)}{(\mathbf{p}^k, \mathbf{Ap}^k)}$$

$$\mathbf{p}^{k+1} = \mathbf{q} + \beta \mathbf{p}^k$$

enddo

図 2 不完全コレスキー分解前処理つき共役勾配法の計算手順

$$a_{i,j} = \begin{cases} -\frac{dx \cdot dz}{2 \left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x-1,y,z}} \right)}, & j = i - 1 \\ -\frac{dy \cdot dz}{2 \left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x+1,y,z}} \right)}, & j = i + 1 \\ -\frac{dx \cdot dz}{2 \left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x,y-1,z}} \right)}, & j = i - nx \\ -\frac{dy \cdot dz}{2 \left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x,y+1,z}} \right)}, & j = i + ny \\ -\frac{dx \cdot dy}{2 \left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x,y,z-1}} \right)}, & j = i - nx \times ny \\ -\frac{dz \cdot dy}{2 \left(\frac{1}{\lambda_{x,y,z}} + \frac{1}{\lambda_{x,y,z+1}} \right)}, & j = i + nx \times ny \\ \sum_{k=1}^N -a_{i,k}, & j = i \\ 0, & \text{others} \end{cases} \quad (4)$$

ここで、 dx 、 dy 、 dz は x,y,z 方向の要素間の長さを示す。 λ は $10^{-5} \sim 10^5$ の範囲を想定しているため、式 4 から、非零要素の最大値と最小値の比は百億のオーダーとなる。

3.2 不完全コレスキー分解前処理付き共役勾配法

図 2 に ICCG 法の計算手順を示す。本図での k は反復回数、 $\mathbf{x}^k$ は k 反復目の近似解ベクトル、 $\mathbf{r}^k$ は残差ベクトル、 $\mathbf{p}^k$ は探索ベクトル、 $\mathbf{q}$ は前処理の結果を格納するベクトル、 $\mathbf{L}$ 、 $\mathbf{D}$ 、および $\mathbf{U}$ は不完全コレスキー分解で得られた狭義上三角、対角および狭義下三角行列を示す。

低精度浮動小数点数は、行列 $\mathbf{L}$ 、 $\mathbf{D}$ 、 $\mathbf{U}$ およびベクトル $\mathbf{r}$ 、 $\mathbf{q}$ に適用した。CG 法の反復内で使用する $\mathbf{A}$ に低精度浮動小数点数を適用した場合、係数行列の条件数によっては解くべき問題そのものが変化してしまう場合があり、また、探索ベクトル $\mathbf{p}$ や残差ベクトル $\mathbf{r}$ を低精度化した場合には、内積の精度悪化によって収束性が大きく変化する可能性がある。対して、CG 法の前処理は、簡易的に $\mathbf{Aq} = \mathbf{r}$ を解ければよく、また CG 法全体に対して半分以上の計算時間を占めるため、低精度化が適している部分である。よって、本研究では前処理部で使用する行列およびベクトルのみに低精度浮動小数点数を適用する。

IC 前処理の並列化には多色順序付けによる手法を適用

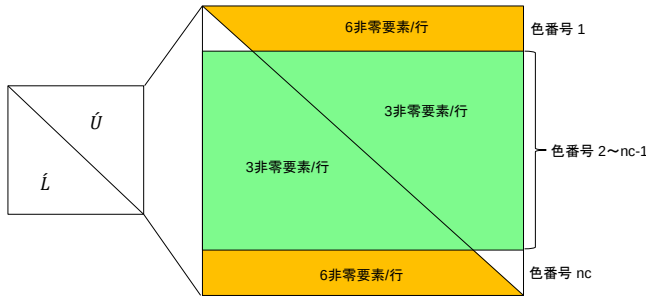


図 3 色分け後の非零分布

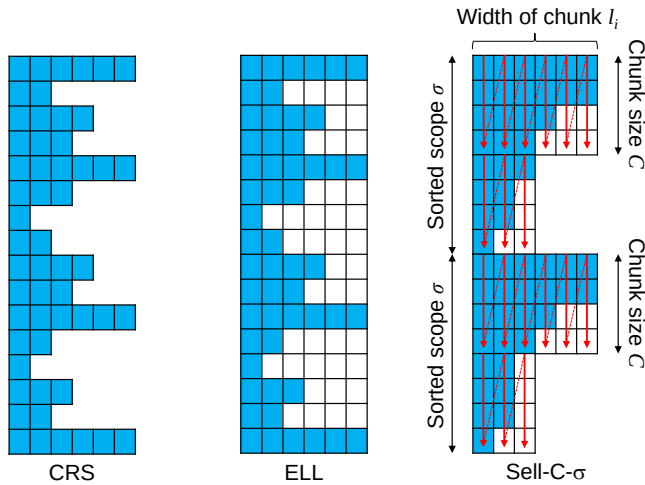


図 4 CRS、ELL、Sell-C- σ の格納形式の違い

している。本研究で適用した多色順序付け法は、Reverse Cuthill-McKee で並び替えを行ったのちに、Cyclic Multi-coloring で多色順序付けを行う CM-RCM(nc) である [10]。CM-RCM(nc) 法は色数の指定が可能であり、また収束性も比較的良好という利点がある。なお、本研究で使用する問題では、nc 色の CM-RCM の適用により、図 (3) のように、 $\hat{L}$ 、 $\hat{U}$ の各行に含まれる非零要素数は 0、3 または 6 となる。

4. 疎行列格納方式

本節では ICCG 法で使用している疎行列格納形式について述べる。本研究で扱う係数行列は疎行列であるため、CRS 格納形式の利用が一般的である。ただし、CRS では効果的な SIMD 演算の利用が難しい。本研究では効果的な SIMD 演算が可能な、ELL および Sell-C- σ 形式の実装も行っている。本節では、これら ELL および Sell-C- σ 形式での格納方法と、その違い (図 4) について述べる。

ELL 形式では、最も非零要素数の多い列に合わせて他の列の要素を 0 パディングで埋めて均等にする。結果、形式的には 2 次元配列のような形になり、列方向または行方向への SIMD 化が容易となる。加えて、本実装の場合は色毎に行方向の要素数が決まっており、かつ同数であるため、ELL 内に 0 パディングの箇所はなく、理想的な状態となる。サンプルコード 1 に ELL 形式での IC 前処理の前進代

サンプルコード 1 ELL のサンプルコード

```
do ic= 2, nc-1
  ip1= (ip-1)*L + ic
  do k= 1, 3
    !$omp simd
    do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      q(i) = q(i) - AL_pre(i,k) * Q(itemL(i,k))
    enddo
  enddo
  !$omp simd
  do i= SMPindex(ip1-1)+1, SMPindex(ip1)
    q(i) = q(i) * Dinv(i)
  enddo
  !$omp barrier
enddo
```

入を示す。サンプルコード上の SIMD 化方向は行方向としている。また、3.2 節で述べたように各色番号での列毎の要素数が決まっているため、サンプルコード 1 では、色番号が 2~nc-1 の場合 (列方向の要素数は 3) を示している。ic は色番号を示しており、各色の計算完了後に barrier 同期を取っている。

次に、Sell-C- σ 形式について述べる。Sell-C- σ 形式は、Moritz Kreutzer らによって提案された格納形式であり [9]、ELL と比較して、より不要な 0 パディングを減らしつつ、SIMD 演算も効果的に利用可能な格納形式である。Sell-C- σ では、Sorting scope σ と Chunk サイズ C という 2 つのパラメータがある。Sorting scope の範囲内では、列方向の非零要素数が少ない順に行方向の並び替えを行い、Chunk サイズの範囲では列方向の非零要素数が同じになるように 0 パディングを行う。Chunk サイズは SIMD 化を促進するパラメータとなるため、計算機アーキテクチャに応じて最適なパラメータが異なり、Sorting scope は Chunk サイズ

サンプルコード 2 Sell-C- σ のサンプルコード

```
do ic= 2, nc-1
  iq1= SMPindex((ip-1)*L + ic-1) + 1
  iq2= SMPindex((ip-1)*L + ic)
  ip0= (ip-1)*L + ic
  iq0= (iq1-1)*CHUNK
  do ib = iq1, iq2
    ib0= (ib-iq1)*CHUNK
    do k= 1, 3
      !$omp simd
      do is = 1, CHUNK
        i= iq0 + ib0 + is
        q(i) = q(i) - AL_pre(ib0+is,k,ip0) &
          * q(itemL(ib0+is,k,ip0))
      enddo
    enddo
  enddo
  do is = 1, CHUNK
    i= iq0 + ib0 + is
    q(i) = q(i) * Dinv(i)
  enddo
  !$omp barrier
enddo
```

表 2 システム仕様

Specifications		Oakforest-PACS	Oakbridge-CX	Wisteria/BDEC-01 Odyssey
CPU	Model	Xeon Phi 7250 (Knights Landing)	Xeon Platinum 8280 Cascade Lake	A64FX
	Number of cores	68	56 (2 Sockets)	48
	Clock	1.4GHz	2.7GHz	2.2GHz
	L1-cache	32kB/core	32kB/core	64kB/core
	L2-cache	1MB/2core	1kB/core	8MB/CMG
Memory	Technology	MCDRAM	DDR4	HBM2
	Size	16GB	192GB	32GB
	Stream triad	495GB/sec	282GB/sec	840GB/sec

表 3 使用したコンパイラおよびオプション

	Compiler	Version	Options
Oakforest-PACS	ifort	19.1.3.304	-O3 -xMIC-AVX512 -qopenmp -align array64byte -ipo
Oakbridge-CX	ifort	19.1.3.304	-O3 -xHost -qqopenmp -align array64byte -ipo
Wisteria/BDEC-01 Odyssey	frtpx	4.5.0 tcscd-1.2.31	-O3 -Kfast,openmp,zfill,A64FX,ARMV8_A

決定後に、より 0 パディングの数が少なくなるパラメータを選ぶのが良い。サンプルコード 2 に Sell-C- σ での IC 前処理の前進代入を示す。CHUNK 変数が Chunk サイズを示しており、SIMD 幅がこのパラメータで決まっている。なお、ELL の項でも述べたように各行の非零要素数は決まっているため、Sorting scope $\sigma = 1$ としており、ELL 同様に 0 パディングはない。

Sell-C- σ の実装では、サンプルコード 2 を基準 (Base) として、より optimize された実装として、k ループを simd 化ループ内に入れた opt1 実装と、k ループを完全に展開した opt2 実装を用意し、これらの評価も行った。

5. 性能評価

本節では低精度浮動小数点数の利用による計算時間短縮の効果を検証した結果を示す。評価結果は、各浮動小数点数の型で最も高速な格納形式を確認したのち、その格納形式を使用して、低精度浮動小数点数の利用による収束性への影響、および計算時間短縮の効果を示す。最も高速な格納形式の評価では、はじめに Sell-C- σ の実装による性能の差を示したのちに、最速な Sell-C- σ の実装を使用して各システム上での Chunk サイズによる計算時間の差を示す。その上で、CRS、ELL と最適な Chunk サイズを使用した、Sell-C- σ の各システム上での計算時間の差を示す。

5.1 評価環境

本評価で使用したシステムは JCAHPC が運営する Oakforest-PACS(OFP) および、東京大学情報基盤センターが運営する Oakbridge-CX(OBCX)、Wisteria/BDEC-01 Odyssey(WO) である。各システムで使用したコンパイラとそのバージョンおよびコンパイルオプションは表 3 に示す。なお、Oakforest-PACS および Oakbridge-CX では、FP16 のハードウェア的なサポートがないため、評価から

除外している。

表 2 に各システムの仕様を示す。本評価で使用した問題の領域は全方向の長さが 1 の正方領域であり、自由度は $256^3 \simeq 16.8\text{M}$ 要素とした。ICCG 法に関しては、IC 前処理以外は全て倍精度で実装しており、前処理部の行列及びベクトルの浮動小数点数の精度を変化させて評価した。また、行列の格納形式の違いは前処理部だけでなく、行列ベクトル積部にも適用している。ICCG の法の収束条件は $\|r^k\|_2/\|r^0\|_2 \leq 10^{-8}$ とし、条件を満たした場合に計算終了とした。

5.2 疎行列格納方式の計算時間への影響

本節では、 λ は空間内で均一 (係数行列は良条件) とし、各システム上での浮動小数点数の型および行列の格納形式の計算時間短縮への効果のみを評価した結果を示す。評価で使用した行列およびベクトルの浮動小数点数の型の組合せは、“行列-ベクトル”で示している。例えば、FP64-FP64 は行列、ベクトルともに FP64 での、FP16-FP32 は行列を FP16、ベクトルを FP32 での格納を示している。以降の図でも同様の表記にて、使用している浮動小数点数の型を示す。なお、ベクトルに FP16 を使用した場合は収束しなくなることを確認しており、評価からは除外している。

はじめに、Sell-C- σ の実装による性能の差を図 5 に示す。図に示した計算時間は前処理部のみを示しており、結果は WO のものを示している。図 5 の結果では、opt2 が最も高速な実装であり、低精度浮動小数点数の使用による計算時間の効果が大きく出ている。低精度浮動小数点数の利用により、計算時間を短縮させるためには、低精度化した状態でも十分にメモリバンド幅律速となるよう、効率的な SIMD 演算の利用が必要である。Base 実装は最内ループの演算量が少ないために、SIMD の効果を得にくく、特に Chunk サイズが小さい場合には他と比較して長い計算時間を要し

ている。一方で、opt1 や opt2 は最内ループの演算量がある程度あるため、Base よりも高速に計算できており、また Chunk サイズが小さい条件でも Base ほど遅くはならない。特に opt2 はループ展開によってソフトウェアパイプラインが効果的に働いており、もっとも高速となった。他のシステムでも同様の傾向が見られ、opt2 が最も高速であったため、以降の SCS の評価結果では opt2 のみを示す。

図 6、7、8 にそれぞれ OFP、OBCX、WO 上で Sell-C- σ opt2 の浮動小数点数の型と Chunk サイズを変更した場合の計算時間への影響を示す。また、表 4 に各システムでの Chunk サイズ 8 に対する最適なサイズでの計算時間の短縮率を示す。図 6～8 から分かるように、システムによって最適な Chunk サイズや、Chunk サイズが計算時間に与える影響が異なっている。OFP での最適な Chunk サイズは 8～16 と小さく、Chunk サイズが大きくなるほど計算時間が長くなる傾向を示している。OBCX では Chunk サイズに対する計算時間の変化はどのシステムよりも小さく、最適な Chunk サイズでも Chunk サイズ 8 に対して 1.3% 程度の計算時間短縮となった。対して、WO では Chunk サイズの違いによる計算時間への影響が他のシステムよりも大きく、また比較的大きな Chunk サイズが最適であることも分かった。特に低精度浮動小数点数を使用している場合に影響が大きく、FP64-FP64 の場合は Chunk サイズ 8 に対して 256 の計算時間は 6.6% の差に対して、FP16-FP32 では Chunk サイズ 512 が最適で、44.6% の差となった。これは、浮動小数点数の低精度化により、SIMD1 命令中に計算可能なデータ数が増えるため、より長い SIMD 長が必要になったとためと考えられる。なお、同程度のメモリアクセス量となる FP16-FP64 と FP32-FP32 の比較では、FP16-FP64 が遅くないが、これは計算時の浮動小数点数の型が、より高精度な方に寄せられるためである。FP32-FP32 は FP32 で、FP16-FP64 は FP64 で計算されるために、1 命令中に計算できる要素数が FP16-FP64 の方が少ないために、

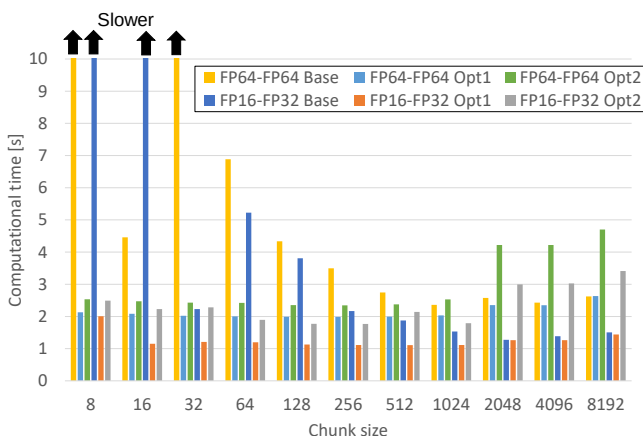


図 5 Sell-C- σ の実装および Chunk サイズに対する計算時間の変化 (データ型は行列が FP16、ベクトルは FP32)

FP32-FP32 の方が高速となった。

図 9 には、異なる浮動小数点数型、疎行列格納方式での前処理部の計算時間を各システム上で計測した結果、図 10 には ICCG 法全体の計測結果を示す。なお、Sell-C- σ (図中では SCS と表記) の結果は各システムでの最適な Chunk

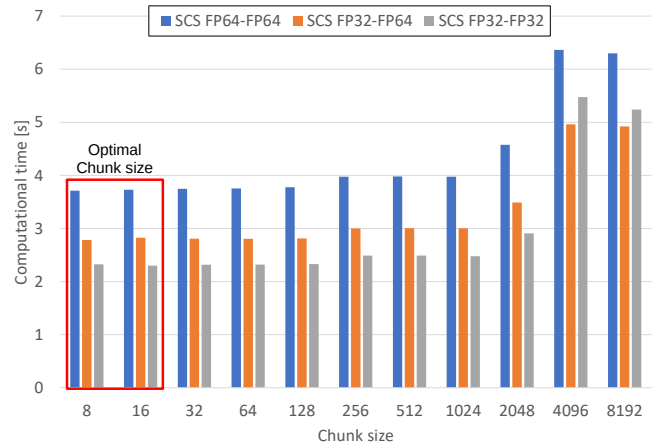


図 6 Oakforest-PACS での各データ型の Chunk サイズに対する計算時間の変化

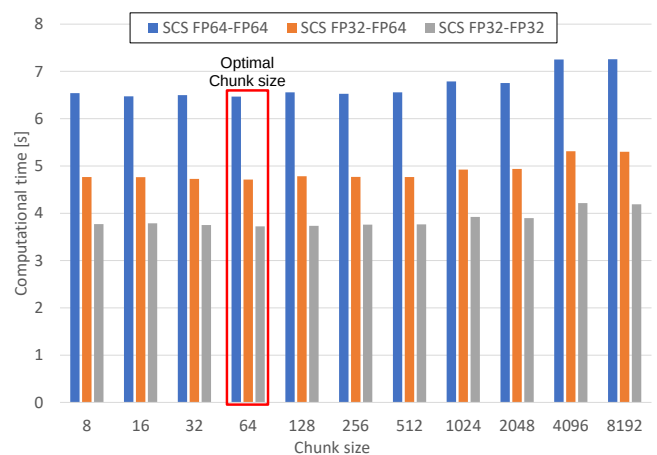


図 7 Oakbridge-CX での各データ型の Chunk サイズに対する計算時間の変化

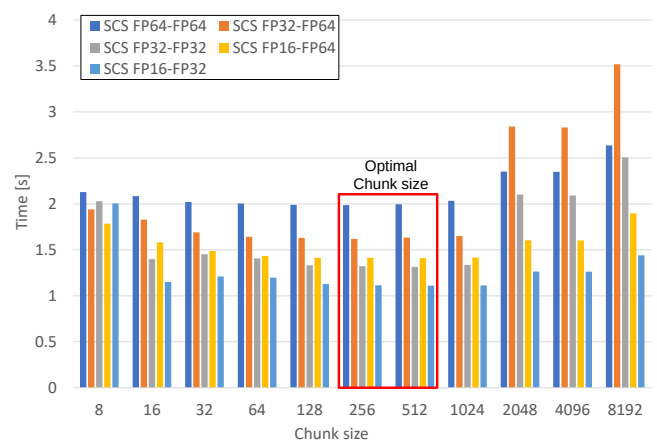


図 8 Wiseria BDEC-01/Odyssey での各データ型の Chunk サイズに対する計算時間の変化

表 4 各システムでの Chunk サイズ 8 に対する最適な Chunk サイズでの計算時間短縮率 [%]

	Chunk サイズ 8 に対する計算時間短縮率 [%]				
	FP64-FP64	FP32-FP64	FP32-FP32	FP16-FP64	FP16-FP32
Oakforest-PACS	0.0	0.0	1.0	none	none
Oakbridge-CX	1.1	1.1	1.3	none	none
Wisteria/BDEC-01 Odyssey	6.6	16.6	35.2	21.0	44.6

サイズを適用している。OBCX ではいずれの疎行列格納形式でも大きな差はなく、CRS 形式では全てのシステムの中で最速となった。OFP や WO では Sell-C- σ の効果が大きく、CRS と比較してそれぞれ FP64-FP64 で、69.3%、79.5%を、FP32-FP32 で 80.8%、88.5%の計算時間短縮を達成した。また ICCG 法全体の計算時間では格納形式の違いによる計算時間の差がさらに大きくなっている。IC 前処理部での低精度浮動小数点数の利用による効果は、高速な疎行列格納形式ほど、計算時間短縮の効果も大きくなっている。これは、Sell-C- σ の最適な実装での考察同様、より効率的な SIMD 演算の利用ができており、メモリアクセス量の削減が大きく寄与したためである。具体的には、Sell-C- σ での FP32-FP32 は FP64-FP64 に対して、OFP では 33.8%、OBCX では 42.4%、WO では 38.0%の、WO での FP16-FP32 は 44.1%の計算時間短縮を達成した。ICCG 法全体の計算時間に対する低精度浮動小数点数の効果は、適用範囲が限定的であるため前処理部での評価と比較して小さくなっている。

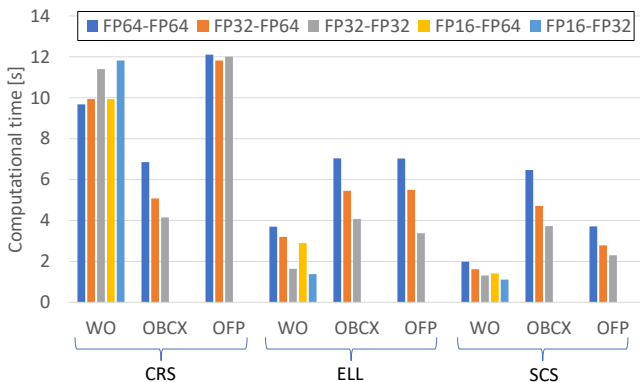


図 9 各システムでの前処理部の計算時間の比較

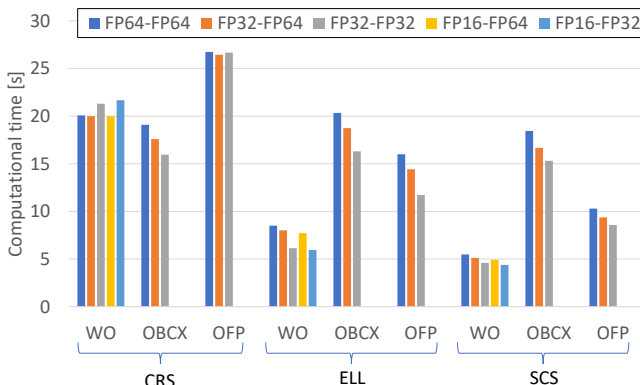


図 10 各システムでの ICCG 法全体の計算時間の比較

5.3 問題の条件に対する低精度浮動小数点数利用の効果

次に、空間内の λ の値を変化させた場合の各浮動小数点数での反復回数および計算時間を示す。評価に使用した問題では、図 11 に示すように、空間内の一部の質量密度 λ_2 を他の部分の質量密度 λ_1 に対して $1 \sim 10^{10}$ まで変化させた。これは、 $\lambda_1 = 10^{-5}$ とし、 λ_2 を $10^{-5} \sim 10^5$ とする条件と同義である。なお、係数行列の格納形式はいずれのシステムでも最も高速であった Sell-C- σ を使用しており、格納形式の違いは収束性に影響しないことを予め確認している。

図 12、13、14 に OFP、OBCX、WO 上での、低精度浮動小数点数の利用による収束性への影響を、図 15～図 17 に計算時間短縮の効果を示す。

収束性の観点では、いずれのシステムでも同様の傾向を示している。若干の違いは使用コンパイラや並列度の違いによるものと推測する。係数行列が良条件の範囲ではどの浮動小数点数でも収束性に差はないが、 λ_2/λ_1 が 20 の辺りから、ベクトルの浮動小数点数を FP32 にした影響が現れはじめ、ベクトルが FP64 の場合と比較して収束性が悪化している。 λ_2/λ_1 が 10^5 を超えると、行列に FP16 使用した場合に収束しなくなっている。これは FP16 の指数部の表現範囲 (表 1) を超えたためである。よって、条件数が少しでも悪くなる要因がある場合にはベクトルの浮動小数点数の型を、全体の計算時間などを考慮して、慎重に選ぶ必要があり、行列側は非零要素の値がその浮動小数点数で表現可能かどうかだけで選んでよいことが分かった。

計算時間の観点でも、システム毎の違いは小さく、係数が良条件の範囲では、低精度の積極的な利用が計算時間短縮に寄与しており、実際に、 $1.0 < \lambda_2/\lambda_1 < 20.0$ の範囲では、FP32-FP32 は FP64-FP64 よりも OFP および OBCX 上で 16.8%、WO 上で 16.5%、FP16-FP32 は WO 上で 20.3%の計算時間短縮を達成した。20.0 < λ_2/λ_1 からは、条件によって最速な浮動小数点数が変化している。最速な浮

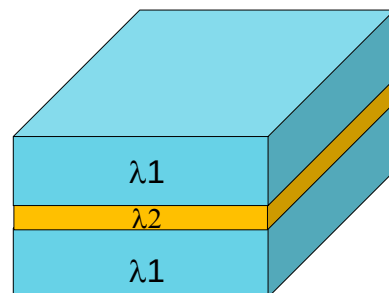


図 11 係数 λ を変化させる評価に使用した問題の形状

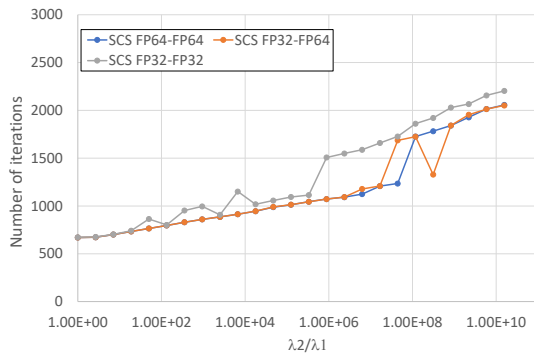


図 12 OFP 上での係数 λ の収束性への影響

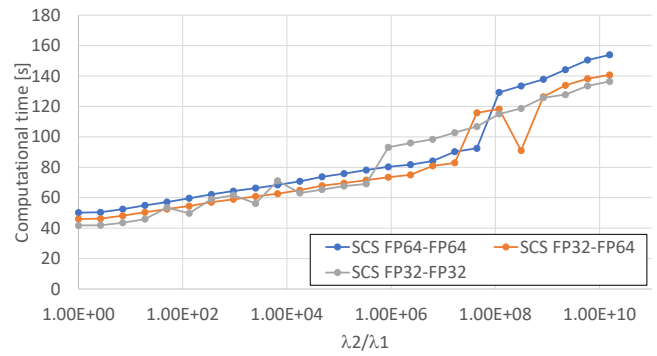


図 15 OFP での係数 λ に対する計算時間短縮の効果

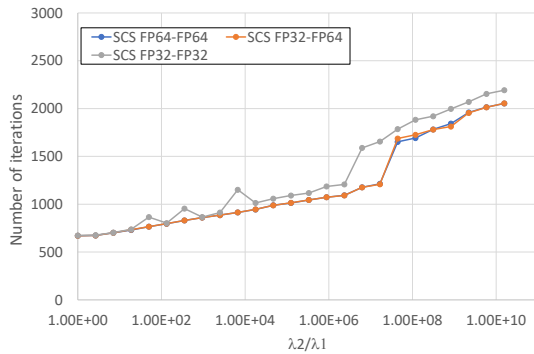


図 13 OBCX 上での係数 λ の収束性への影響

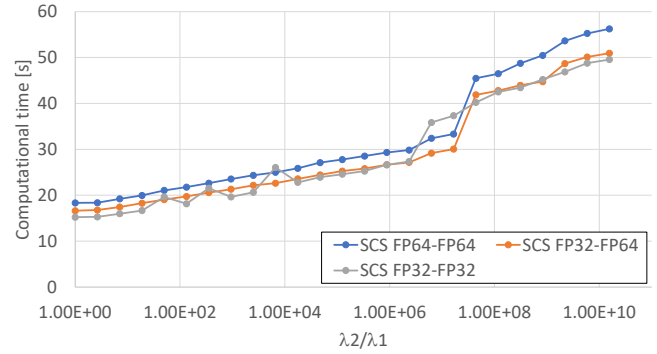


図 16 OBCX での係数 λ に対する計算時間短縮の効果

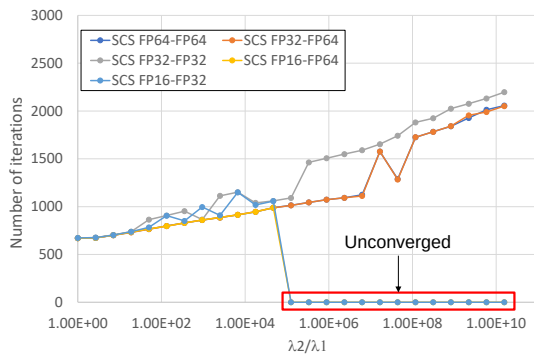


図 14 Wisteria/BDEC-01 Odyssey 上での係数 λ の収束性への影響

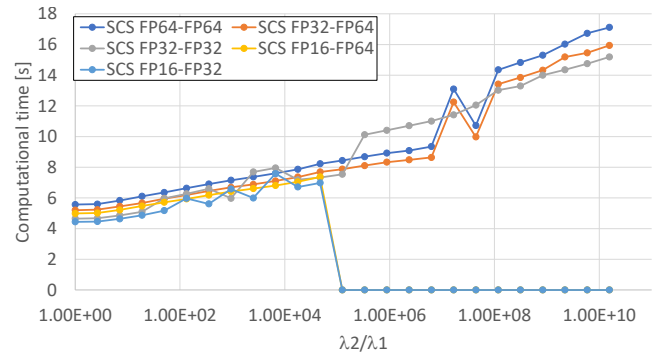


図 17 Wisteria/BDEC-01 Odyssey での係数 λ に対する計算時間短縮の効果

動小数点数の組み合わせを、 $\lambda_2/\lambda_1 = 1$ の条件から列挙すると、最速な条件は、FP16-FP32 (WO のみ)、FP32-FP32、FP32-FP64、FP32-FP32 という傾向を示している。最後に FP32-FP32 が最速となっているのは、ベクトルが FP64 でも、収束性が悪化し、FP32 の場合に近づいているためである。

6. まとめ

本研究では pHEAT-3D アプリケーションを、FVM を用いて離散化した結果得られる連立一次方程式を解く ICCG 法の、IC 前処理に低精度浮動小数点数を適用し、計算時間短縮の効果を評価した。評価のためには低精度浮動小数点数を使用した場合でも、十分なメモリアクセスが必要となるため、効果的な SIMD 演算が利用可能な Sell-C- σ で行列

を格納し、CRS や ELL との比較も行った。係数行列が良条件の範囲内で、疎行列の格納形式および浮動小数点数型の計算時間短縮への効果を評価した結果、Sell-C- σ のような SIMD 演算を効果的に利用可能な格納形式では低精度浮動小数点数の効果を得られた。全体の計算時間の観点では、問題の条件によって最速な浮動小数点数の組み合わせは異なるが、良条件の範囲では、低精度浮動小数点数である FP16 や FP32 利用の確認できた。具体的には $1.0 < \lambda_2/\lambda_1 < 20.0$ の範囲では、FP32-FP32 は FP64-FP64 よりも OFP および OBCX 上で 16.8%、WO 上で 16.5%、FP16-FP32 は WO 上で 20.3% の計算時間短縮を達成した。

今後の研究として、FP21[11] や FP42 といった IEEE754 に規定されていない浮動小数点数型の利用の検討や、最速な型を自動的に選択する自動チューニングなどを行って

いく。

謝辞 本研究の遂行に関しては、本研究は JSPS 科研費 JP19H05662 の助成および、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・インフラからの計算資源の支援を受けている (課題番号:jh200037-NAH)。この場を借りて感謝の意を表する。

参考文献

- [1] Haidar, A., Tomov, S., Dongarra, J. and Higham, N. J.: Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers, *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 603–613 (2018).
- [2] Lee, J., Lee, J., Han, D., Lee, J., Park, G. and Yoo, H.: 7.7 LNPU: A 25.3TFLOPS/W Sparse Deep-Neural-Network Learning Processor with Fine-Grained Mixed Precision of FP8-FP16, *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, pp. 142–144 (2019).
- [3] Matsuoka, N., Qiu, L., Li, X., Omori, T. and Hashimoto, K.: Applicability of single precision graphics processing unit for fast simulation of 2D surface acoustic wave devices using an hierarchical cascading technique, *Japanese Journal of Applied Physics*, Vol. 58, No. SG, p. SGGC11 (2019).
- [4] Fujita, K., Horikoshi, M., Ichimura, T., Meadows, L., Nakajima, K., Hori, M. and Maddegadara, L.: Development of element-by-element kernel algorithms in unstructured finite-element solvers for many-core wide-SIMD CPUs: Application to earthquake simulation, *Journal of Computational Science*, Vol. 45, p. 101174 (2020).
- [5] Hess, B.: P-LINCS : A Parallel Linear Constraint Solver for Molecular Simulation, *Journal of Chemical Theory and Computation*, Vol. 4, No. 1, pp. 116–122 (2008).
- [6] Sakamoto, R., Kondo, M., Fujita, K., Ichimura, T. and Nakajima, K.: The Effectiveness of Low-Precision Floating Arithmetic on Numerical Codes: A Case Study on Power Consumption, *HPCAsia2020*, p. 199–206 (2020).
- [7] 研吾中島, 龍一坂本, 哲也星野, 英志有間, 敏博 埴, 正章近藤: 低精度演算とアプリケーション性能, 技術報告 5 (2020).
- [8] Bell, N. and Garland, M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors, *Proceedings of the conference on high performance computing networking, storage and analysis*, pp. 1–11 (2009).
- [9] 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*, Vol. 36, No. 5, pp. C401–C423 (2014).
- [10] Washio, T., Maruyama, K., Osoda, T., Shimizu, F. et al.: Efficient implementations of block sparse matrix operations on shared memory vector machines, *SNA 2000. The fourth international conference on supercomputing in nuclear applications* (2000).
- [11] Yamaguchi, T., Fujita, K., Ichimura, T., Naruse, A., Lalith, M. and Hori, M.: GPU implementation of a sophisticated implicit low-order finite element solver with FP21-32-64 computation using OpenACC, *Lecture Notes in Computer Science, WACCPD 2019*, Vol. 12017 (2019).