Vol.2013-HPC-139 No.7
2013/5/29

CUMP を用いた多倍長精度演算による Krylov 部分空間解法の GPU による高速化

廣川 祐太[1,a)]　藤田 宜久[2]　伊東 拓[1]　生野 壮一郎[1]

概要：大規模疎行列を係数行列とする連立 1 次方程式の求解には，Krylov 部分空間解法が有効な手法として知られている．同手法の反復回数は，各反復において生じる誤差に依存する．そのため，多倍長精度演算を用いて高精度に計算を行えば，誤差が減少し反復回数が減少する．しかしながら，多倍長精度演算は倍精度演算や単精度演算に比べメモリ使用量や計算量が多く，GPGPU 等による高速化が研究されている．反復回数や精度改善には前処理が用いられるが，一般に前処理行列の生成には時間を要してしまうことから可変的前処理が注目されている．また，可変的前処理部分は高精度である必要がなく，倍精度演算，単精度演算の計算精度でも十分であると考えられる．本研究では，CUMP を用いて多倍長精度混合精度型可変的前処理付き Krylov 部分空間解法を GPU 上に実装し，CPU による解法に比べて最大 19.3 倍の高速化を達成した．

1. はじめに

物理的，工学的現象を定式化し，有限要素法や差分法を用いて離散化することで得られる連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の係数行列 A は一般的に大規模疎行列となる．Krylov 部分空間解法は，大規模疎行列や特異行列を係数行列にもつ連立 1 次方程式に対して有効な解法である．また，同解法は行列ベクトル積，内積，ベクトルの乗加算で構成されており，非常に並列化が容易であるという特徴をもつ．そのため，GPU を用いて高速かつ高精度に求解する研究が精力的に行われている [1], [2].

一般に，Krylov 部分空間解法の反復回数は，各反復内の計算で起こる丸め誤差や打ち切り誤差の蓄積に依存していることが知られている．すなわち，多倍長精度演算等を用いて高精度に計算を進めれば反復回数の減少が期待できる．一般的には GNU Multiple Precision Arithmetic Library (GMP) [3] 等のライブラリを用いて多倍長精度演算を実現するが，GPU 上で多倍長精度演算を実現するために GMP との協調計算が可能な CUDA Multiple Precision Arithmetic Library (CUMP) があり，GPU への実装方法等の研究が行われている [4], [5]．しかしながら，多倍長精度演算は倍精度演算や単精度演算に比べメモリ使用量が飛躍的に増加してしまうため，1 台あたりのメモリ搭載量が少ない GPU では，演算規模の面で大きな制約を受けてしまう．この問題の解決方法の 1 つとして，GPU クラスタを用い，複数の GPU を用いることで，総メモリ領域の増加を図り，さらなる高速化を図る．

本研究では，GPU クラスタである筑波大学の HA-PACS を用いて，CPU による多倍長精度演算として GMP を，GPU による多倍長精度演算として CUMP を用い，多倍長精度演算を用いた Krylov 部分空間解法を実装することで，様々な方法で得られる連立 1 次方程式に対しより精度の高い解を高速に得ることを目的とする．

2. 多倍長精度演算

倍精度演算は，IEEE 754 の定義に従うと 10 進数にして約 16 桁程度の精度しかもつことができず，それ以上の精度を要求する問題が発生した際，その精度は保証されない．そのため，倍精度よりも高い精度の演算を行う研究が行われてきた．特に代表される多倍長精度演算の実装として GMP，MPFR がある [6].

多倍長精度演算は，ハードウェアによる実装が容易な固定精度である倍精度演算や単精度演算とは異なり，変数のメモリ管理や演算などすべてソフトウェアにより実装されているため，倍精度演算や単精度演算に比べて，メモリ使用量の増加や 1 回の演算にかかる計算量の増加など，大幅な演算性能の低下が見込まれる．そのため，多倍長精度演算を用いる場合には高速化を行うことは必須である．GPU 上で多倍長精度演算を実現するライブラリとして，GMP との協調計算が可能な CUMP と，ARPREC をもとにした

[1] 東京工科大学コンピュータサイエンス学部
[2] 名古屋大学工学研究科エネルギー理工学専攻
[a)] hirokawa@nal.ikulab.org

Let $\boldsymbol{x}_0$ be an initial guess.
$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Roughly solve $A\boldsymbol{z}_0 = \boldsymbol{r}_0$ using some iterative method
$\boldsymbol{p}_0 = \boldsymbol{z}_0$
for $k = 0, 1, ...,$ until $||\boldsymbol{r}||_2/||\boldsymbol{b}||_2 \leq \varepsilon$
  $\alpha_k = \frac{(\boldsymbol{r}_k, \boldsymbol{z}_k)}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)}$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$
  Roughly solve $A\boldsymbol{z}_{k+1} = \boldsymbol{r}_{k+1}$
    − using some iterative method
  $\beta_k = \frac{(\boldsymbol{r}_{k+1}, \boldsymbol{z}_{k+1})}{(\boldsymbol{r}_k, \boldsymbol{z}_k)}$
  $\boldsymbol{p}_{k+1} = \boldsymbol{z}_{k+1} + \beta_k \boldsymbol{p}_k$
end for

図 **1** VPCG 法のアルゴリズム.

GARPREC [5] などが考案されている.

前述したように，CUMP は GMP をもとにした CUDA による多倍長精度演算を提供するライブラリで，GMP との間でデータ転送ができ協調計算を可能としている．しかしながら，CUMP に実装されている演算法則は加減算と乗算のみであり，複雑な計算を必要とするコードを実装するのは困難である．本研究では，疎行列ベクトル積や内積の計算を行うため，総和演算を実現する必要がある．その際，総和結果を格納する変数をゼロで初期化する必要があるが，CUMP には GPU カーネル内で倍精度数値や整数値などの組み込み型を代入する関数がなく，総和演算を容易には実現できない．そのため，本研究では多倍長精度変数のデータを初期化状態にする関数を新たに用意することで，ゼロ初期化のみを独自に実装した.

本研究では，GPU クラスタのノード間の通信には MPI を用いる．しかしながら，GMP は MPI などで直接転送できるようなデータ構造をとっておらず，MPI で GMP データを送受信するためにはプログラム上でデータをシリアライズしてから転送する必要がある．GMP のデータ構造はドキュメント化されており，データ構造を参照して適切にデータをシリアライズすることが可能である [7].

3. 混合精度型可変的前処理

従来，Krylov 部分空間解法で反復回数や精度改善のために前処理付き解法が使用されており，前処理行列の生成には不完全 LU 分解法や不完全コレスキー分解法が用いられる [8]．しかしながら，前処理付き解法は前処理行列の構築に反復計算よりも多くの時間を費やし，また前処理そのものも後退代入で行われるため並列化が困難である．そのため，前処理行列の構築を必要としない可変的前処理が注目されている．可変的前処理付き (VP, Variable Preconditioned) CG 法のアルゴリズムと VPGCR (m) 法のアルゴリズムを 図 **1**，図 **2** に示す.

可変的前処理は任意の反復法を用いて反復毎に $A\boldsymbol{z}_{k+1} = \boldsymbol{r}_{k+1}$ を計算する．可変的前処理の計算を内部解法と呼び，Krylov 部分空間解法の計算部分を外部解法と呼ぶ．内部解法には，定常反復法等の解法を適用できるため GPU に適した解法を使用可能である．また，同解法には内部解法の相対残差ノルムが各反復において，

$$\frac{||\boldsymbol{r}_{k+1} - A\boldsymbol{z}_{k+1}||_2}{||\boldsymbol{r}_{k+1}||_2} < 1 \tag{1}$$

を満たすならば，外部解法が収束するという収束定理が存在する [9]．この収束定理は，外部解法に残差の単調減少性をもつ Arnoldi 原理に基づく解法である GCR (m) 法を用いることにより導かれている．また，内部解法は反復の初期の残差減少性能が良い定常反復法が用られている．しかしながら，GCR (m) 法はリスタート回数 m の選択によって，メモリ使用量が増大することが考えられ，GPU 上や多倍長精度演算での実装が困難である可能性がある．そのため，本研究では，外部解法を GCR (m) 法以外の Krylov 部分空間解法を用いた VP Krylov 部分空間解法を実装し評価を行う.

(1) よりわかるように，内部解法は多倍長精度演算を用いずとも，倍精度演算や単精度演算の精度範囲で十分に計算可能であると考えられる．前述のように，多倍長精度演算はソフトウェアによる計算を行うため，倍精度演算や単精度演算に比べ大幅な演算性能の低下が見込まれる．そこで本研究では，VP Krylov 部分空間解法において外部解法に多倍長精度演算を用い，内部解法に倍精度演算もしくは単精度演算を用いる混合精度型 VP Krylov 部分空間解法を採用し，GPU クラスタへの実装を行う．また，外部解

Let $\boldsymbol{x}_0$ be an initial guess.
loop
  $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
  Roughly solve $A\boldsymbol{p}_0 = \boldsymbol{r}_0$ using some iterative method
  $\boldsymbol{q}_0 = A\boldsymbol{p}_0$
  for $k = 0, 1, ..., m-1$ until $||\boldsymbol{r}_k||_2/||\boldsymbol{b}||_2 \leq \varepsilon$
    $\alpha_k = \frac{(\boldsymbol{r}_k, \boldsymbol{q}_k)}{(\boldsymbol{q}_k, \boldsymbol{q}_k)}$
    $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
    $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{q}_k$
    Roughly solve $A\boldsymbol{z}_{k+1} = \boldsymbol{r}_{k+1}$
      − using some iterative method
    $\boldsymbol{p}_{k+1} = \boldsymbol{q}_{k+1} = 0$
    for $i = 0, 1, ..., k$
      $\beta_i = -\frac{(A\boldsymbol{z}_{k+1}, \boldsymbol{q}_i)}{(\boldsymbol{q}_i, \boldsymbol{q}_i)}$
      $\boldsymbol{p}_{k+1}^{i+1} = \boldsymbol{p}_{k+1}^{i} + \beta_i \boldsymbol{p}_i$
      $\boldsymbol{q}_{k+1}^{i+1} = \boldsymbol{q}_{k+1}^{i} + \beta_i \boldsymbol{q}_i$
    end for
    $\boldsymbol{p}_{k+1} = \boldsymbol{z}_{k+1} + \boldsymbol{p}_{k+1}^{k+1}$
    $\boldsymbol{q}_{k+1} = A\boldsymbol{z}_{k+1} + \boldsymbol{q}_{k+1}^{k+1}$
  end for
  $\boldsymbol{x}_0 = \boldsymbol{x}_m$
end loop

図 **2** VPGCR (m) 法のアルゴリズム.

表 1 実験環境.

CPU	Intel E5 2.6GHz ×2
CPU Memory	128GB
CPU Peak	332.8GFLOPS/Node
GPU	NVIDIA M2090 ×4
GPU Memory	24GB/Node
GPU Peak	2660GFLOPS/Node
Infiniband	Mellanox Connect-X3 dual head
CUDA	ver 4.2.9
GMP	ver 5.0.5
CUMP	ver 1.0.1
MVAPICH2	ver 1.8.1
CPU Compiler	icc 13.0 (-O3 -march=native)
GPU Compiler	nvcc 4.2 (-O3 -arch=sm_20)

```
tid = blockDim.x × blockIdx.x + threadIdx.x
if tid < n then
  s = 0
  for i = 0, 1, ..., nz[tid]
    j = jd[i] + tid
    s = s + mat[j] × vec[ col[j] ]
  end for
  out[ perm[tid] ] = s
end if
```

図 3 GPU による JDS の疎行列ベクトル積のアルゴリズム．ただし，n は行列の次元数，vec，out はベクトル，mat は非零要素，nz は 1 行に含まれる非零要素数，jd は転値後の各行の先頭要素の位置，col は行列の列番号，perm は並び替え前の行番号とする．

法に多倍長精度演算を用いた GCR (m) 法および CG 法を実装し，様々な係数行列をもつ連立 1 次方程式に対する評価を行う．

4. 数値実験

評価には，筑波大学計算科学研究センターの HA-PACS ベースクラスタの 8 ノードを用いる．HA-PACS はピーク性能にして 802TFLOPS の計算能力をもつ密結合型 GPU クラスタである [10]．HA-PACS では 1 ノードに対して NVIDIA 社の M2090 が 4 台搭載されており，GPU1 台のピーク性能は 665GFLOPS である．計算ノード 1 台のピーク性能は，CPU および GPU を合わせて 3TFLOPS になる．また各ノード間は Infiniband で接続されており，MPI による通信を高速に行うことが可能となっている．

HA-PACS 計算ノードの仕様および本研究で使用するソフトウェアとバージョンなど，実験環境を 表 **1** に示す．MVAPICH2 は Infiniband に最適化された MPI 実装を提供しており，CUDA のメモリ内データを MPI を介して直接他のノードへ送受信できる Remote Direct Memory Access (RDMA) を独自に提供しているが，多倍長精度演算を用いた解法では GMP および CUMP を用いるため使用できない [11]．本研究では，並列化手法として 1 ノードによる CPU 並列化 (2-CPU)，8 ノードによる MPI を用いた CPU 並列化 (16-CPU)，1 ノードによる GPU4 台を用いた並列化 (4-GPU)，8 ノードによる MPI および GPU32 台を用いた並列化 (32-GPU) の 4 種類の並列化と逐次実行の計 5 種類で評価を行う．CPU 上の計算の並列化には OpenMP を使用し，ノード間通信には MPI を用いる．また，GMP および CUMP の精度は実用的な桁数として 10 進数 100 桁精度とした．

4.1 疎行列ベクトル積の高速化

図 1，図 2 のアルゴリズムからわかるように，Krylov 部分空間解法は計算の殆どをベクトルの加算演算，ベクトルの内積，行列ベクトル積に費やす．その中でも，行列ベクトル積は計算量が多く，特に高速化を行う必要がある．本研究で取り扱う連立 1 次方程式は係数行列 A が大規模な疎行列となるため，疎行列の特性に合わせて疎行列格納方式を選択する必要がある．

本研究では CPU の疎行列格納形式として Compressed Row Storage (CRS) を使用し，GPU の疎行列格納形式として Jagged Diagonal Storage (JDS) と CRS を用いる [12]．JDS はベクトル計算機向けに考案された格納形式であり，また，一般的にベクトル計算機向けのアルゴリズムは計算のベクトル長が長くなるように設計されているため，GPU との相性が良い [13]．GPU による JDS の疎行列ベクトル積のアルゴリズムを 図 **3** に示す．JDS による疎行列ベクトル積では，1 行の計算を CUDA カーネルの 1 スレッドで計算するため，スレッド間でリダクションが不要であるという利点がある．そのため，1 行に含まれる非零要素数が少ない場合には高速に計算できる．JDS は 1 行に含まれる非零要素数で行の並び替えを行い，column-major で非零要素と列番号を格納する．よって，Half-Warp 内のスレッドは連続したメモリ領域にアクセスするため，メモリアクセス効率が高い．GPU は Warp 単位でスレッドを実行するため，スレッドの処理量は均一である方が望ましい．JDS の場合，各行の非零要素数が多い順に並べ替えられているため，処理量のある程度の均一化が望まれる．また，CRS は GPU による疎行列ベクトル積でよく用いられる形式で，1 行を計算するときに用いる最適なスレッド数を自動チューニングする方法も研究されている [14]．

疎行列ベクトル積の性能評価には，The University of Florida Sparse Matrix Collection から 6 個の疎行列を引用し使用する [15]．評価に使用する 6 個の疎行列を 表 **2** に示し，疎行列ベクトル積の CPU と GPU の性能評価を 表 **3** に示す．FLOPS は非零要素数 N_z，計算時間 t を用いて $2N_z/t$ で表される．1 行に含まれる非零要素数や，行列の疎性にも左右されるが，GPU による JDS の疎行列ベクト

表 2　疎行列ベクトル積の評価に用いた疎行列．
ただし，N は次元数．N_z は非零要素数，N_z/Row は 1 行に含まれる最大の非零要素数である．

Matrix	N	N_z	N_z/Row
circuit5M_dc	3523317	14865049	24
Freescale1	3428755	17052626	25
kkt_power	2063494	12771361	123
G3_circuit	1585478	7660826	6
webbase-1M	1000005	3105536	4700
mark3jac120	54929	322483	44

表 3　疎行列ベクトル積の CPU と GPU の性能比較．
単位は MFLOPS である．ただし，GMP および CUMP の精度を 10 進数 100 桁精度とし，CPU では 2-CPU で CRS を用いた疎行列ベクトル積を実装し，GPU では 4-GPU で JDS および CRS を用いた疎行列ベクトル積を実装している．

Matrix	CPU CRS	GPU JDS	GPU CRS
circuit5M_dc	45.83	761.20	107.88
Freescale1	44.76	619.18	105.38
kkt_power	40.23	361.51	132.21
G3_circuit	42.67	1037.70	124.59
webbase-1M	43.43	55.93	92.74
mark3jac120	46.81	188.59	132.63

ル積は CPU に比べて最大 24.5 倍の高速化ができている．また，GPU による CRS の疎行列ベクトル積は 1 行に含まれる非零要素数が多い場合，JDS と比較して高速に計算できる場合もあることがわかる．この結果から，多倍長精度演算による Krylov 部分空間解法も，GPU による高速化が期待できると云えよう．

4.2　Krylov 部分空間解法の高速化

前節の実験で用いた行列 (表 2) から最も次元数の大きい circuit5M_dc を用いて評価を行う．この行列は 表 3 の結果から，JDS で格納し，疎行列ベクトル積を評価することで，高速に計算できると考えられる．$\boldsymbol{Ax} = \boldsymbol{b}$ での未知ベクトル $\boldsymbol{x}$ と既知ベクトル $\boldsymbol{b}$ の値は，疎行列の非零要素の最大値と最小値を範囲として生成した乱数を $\boldsymbol{x}$ として仮定し，$\boldsymbol{Ax}$ の結果を $\boldsymbol{b}$ とした．

収束判定子を 10^{-16} とし，Krylov 部分空間解法を 4 種類の並列化手法を用いて高速化した場合と逐次実行した場合の計算時間を 図 4 に示す．CG 法を 16-CPU，32-GPU で並列化した際，その殆どは通信時間で占められており，計算時間に比べ通信時間はおよそ 10 倍程度となっている．これは 図 1 や 図 2 のアルゴリズムからわかるように，Krylov 部分空間解法では各反復毎に行列ベクトル積を行う必要がある．そのため，行列ベクトル積に用いるベクトルを全ノード間で共有する必要がある．しかしながら，本研究では GMP を用いているため，1 変数あたりのデータサイズはおおよそ 32Byte 程度であり，次元数が 3523317 で

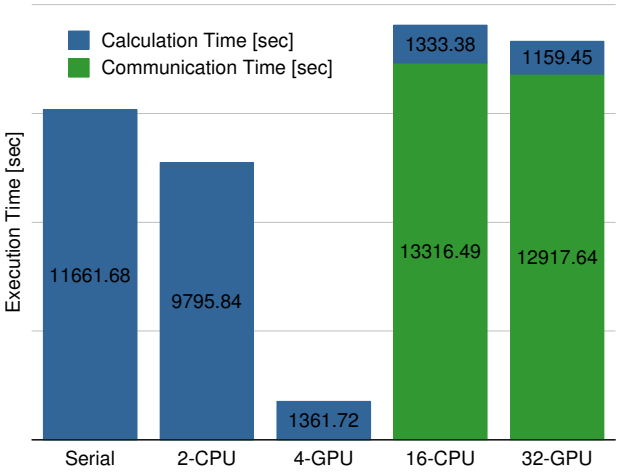

図 4　CG 法の実行時間．ただし，収束判定子を 10^{-16} とし，GMP および CUMP の精度を 10 進数 100 桁精度とする．

表 4　疎行列格納形式を切り替えた際の CG 法の実行時間．
ただし，4-GPU で並列化を行い，収束判定子を 10^{-16} とし，GMP および CUMP の精度を 10 進数 100 桁精度とする．

Type	Calculation Time [sec]
JDS	1361.72
CRS	3734.30

あることからおよそ 100MByte 以上のデータをベクトル共有通信でやりとりしているため，通信時間が大幅に増加している．よって，ベクトル共有通信は計算とオーバーラップさせた方が良いと考えられるが，図 1 および 図 2 からわかるようにベクトル共有通信中に可能な演算が少なく，1 回の内積しか，共有するべきベクトルの通信から行列ベクトル積の間までにできる計算がない．即ち，Krylov 部分空間解法は各反復でベクトルの共有が必要になるため，どのような問題であっても複数ノードによる計算は 1 つのノードを用いる計算よりも遅くなってしまうと考えられる．

前節より，疎行列ベクトル積を毎反復計算する Krylov 部分空間解法では，疎行列格納形式の違いが，高速化に大きく影響を与えると考えられる．4-GPU で並列化し，疎行列格納形式を切り替えた際の CG 法の実行時間を 表 4 に示す．JDS で格納し疎行列ベクトル積を行った場合，CRS で格納した場合に比べて 2.7 倍以上高速化されており，疎行列格納形式の違いが，GPU を用いた Krylov 部分空間解法の計算時間に大きく関わってくることがわかる．

4.3　VP Krylov 部分空間解法の高速化

前節の結果から，4-GPU での計算が最も高速ではあるものの反復回数が多く，高速化率は高いが計算時間は長いという状態になっている．そのため，収束性能改善のために VP Krylov 部分空間解法を実装し，多倍長精度演算による解法のさらなる高速化を図る．

内部解法には Jacobi Over Relaxation (JOR) 法を用いる．JOR 法のアルゴリズムを 図 5 に示す．JOR 法は，対

Let $\boldsymbol{x}_0$ be an initial guess.
for $k = 1, 2, ..., n$
  for $i = 1, 2, ..., n$
    $\sigma = 0$
    for $j = 1, 2, ..., i-1, i+1, ..., n$
      $\sigma = \sigma + \boldsymbol{a}_{ij}\boldsymbol{x}_j^k$
    end for
    $\sigma = (\boldsymbol{b}_i - \sigma) \times (\boldsymbol{a}_{ii})^{-1}$
    $\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^k + \omega(\sigma - \boldsymbol{x}_i^k)$
  end for
  if $||\boldsymbol{x}_{k+1} - \boldsymbol{x}_k||_2 / ||\boldsymbol{x}_{k+1}||_2 \leq \epsilon$ **then**
    break
  end if
end for

図 5　JOR 法のアルゴリズム．ただし，ω は緩和係数である．

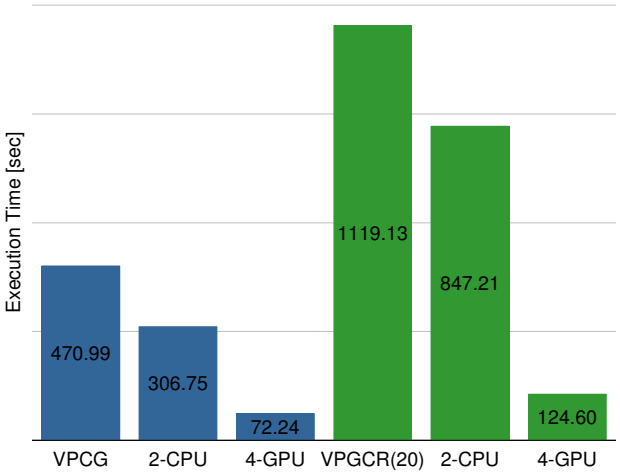

図 6　VP Krylov 部分空間解法の計算時間．ただし，外部解法の収束判定子を 10^{-16} とし，内部解法に用いた JOR 法の緩和係数を 0.6，GMP および CUMP の精度を 10 進数 100 桁精度とする．

角要素が非零である必要があるが，並列化が容易な定常反復法である．しかしながら，JOR 法では対角要素を用いて，CUMP では実装されていない演算法則である除算を行う必要があり，また多倍長精度演算では倍精度演算や単精度演算に比べて除算の性能低下が顕著であるため，対角要素の逆数を計算前に用意する必要がある．

可変的前処理では，内部解法として用いる解法の収束判定子や最大反復回数の最適な値を設定する明確な方法はわかっておらず，外部解法や内部解法の組み合わせによって計算時間や反復回数が変化してしまう．そこで本研究では，VPCG 法の最大内部反復回数を 2，内部収束判定子を 10^{-1} に固定し，VPGCR (m) 法の最大内部反復回数を 6，内部収束判定子を 10^{-2} に固定した．外部解法の収束判定子を 10^{-16}，リスタート回数を 20 とし，JOR 法の緩和係数を 0.6 として VP Krylov 部分空間解法を 2-CPU と 4-GPU で実装した際の計算時間を 図 **6** に示す．VP Krylov 部分空間解法では，収束性能が改善され Krylov 部分空間解法

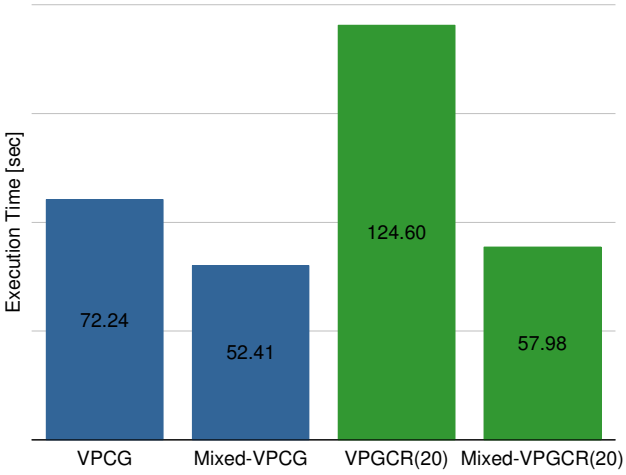

図 7　4-GPU で並列化した混合精度型 VP Krylov 部分空間解法の計算時間．VP Krylov 部分空間解法の計算時間も記載する．ただし，外部解法の収束判定子を 10^{-16} とし，内部解法に用いた JOR 法の緩和係数を 0.6 とし，GMP および CUMP の精度を 10 進数 100 桁精度とする．混合精度型可変的前処理の内部解法には倍精度演算を用いた．

に比べ大幅に反復回数が少なくなることがわかる．高速化率は若干下がるが，4-GPU による並列化を行った VPCG 法は逐次実行に比べ 6.5 倍高速に計算できており，2-CPU による並列化に比べて 4.25 倍高速に計算できた．また，VPGCR (m) 法では 8.98 倍，2-CPU による並列化に比べて 6.79 倍高速に計算できた．

VP Krylov 部分空間解法の可変的前処理部分では，高い精度を必要とせず，多倍長精度演算を用いて計算する必要はないということを既に述べた．可変的前処理の内部解法の精度を倍精度演算として混合精度型 VP Krylov 部分空間解法の高速化を行った際の計算時間を 図 **7** に示す．内部解法を倍精度演算で解くことで，内部解法も多倍長精度演算で解く場合と比べて VPCG 法では 1.32 倍，VPGCR (m) 法では 2.14 倍高速に計算できた．VPGCR (m) 法では，VPCG 法に比べて内部解法の計算回数が増えるため，混合精度型解法による高速化の恩恵が VPCG 法に比べて多く得られたと考えられる．

5. おわりに

本研究では，多倍長精度演算を用いた Krylov 部分空間解法の高速化を目的とした．まずはじめに，Krylov 部分空間解法で最も計算量を占める疎行列ベクトル積の GPU による高速化を評価した．疎行列ベクトル積では，疎行列の形状により，JDS では CPU に比べて最大 24.5 倍の高速化を達成し，CRS では最大 3.21 倍の高速化を達成した．次に，Krylov 部分空間解法として CG 法を実装し，4 種類の並列化と逐次実行をした場合の比較を行い，4-GPU による高速化を行った場合，逐次実行時に比べて 8.56 倍の高速化を達成した．複数ノードで計算した場合，疎行列ベクトル

積に使用するベクトルを共有する必要があるが，通信の際に計算できるのがベクトルの内積演算 1 つのみであり，通信時間を隠蔽できるだけの計算がないため，高速化は困難である．そのため，VP Krylov 部分空間解法は 2-CPU と 4-GPU により並列化を行い，逐次実行時に比べて 4-GPU による並列化により VPCG 法は 6.5 倍の高速化，VPGCR (m) 法では 8.98 倍の高速化を達成した．また，収束定理 (1) から，内部解法の計算は多倍長精度演算でなくとも，倍精度演算や単精度演算でも十分な精度は得られると考え，内部解法を倍精度演算で計算した混合精度型 VP Krylov 部分空間解法を 4-GPU で実装し，逐次実行時に比べ VPCG 法は 8.66 倍，VPGCR (m) 法は 19.3 倍高速化できた．高速化率に差が出たのは，VPCG 法と VPGCR (m) 法では内部解法のパラメータに違いがあり，VPCG 法に比べ最大内部反復回数が多く，内部収束判定子も高精度な VPGCR (m) 法が，VPCG 法に比べ混合精度型解法による恩恵を多く得られたと考えられる．

以上の結果から，GPU による多倍長精度演算は実問題の高速化に対しても非常に有効であると考えられる．しかしながら，これまで用いてきた倍精度演算による解法との比較や，VPCG 法，VPGCR (m) 法以外の VP Krylov 部分空間解法の実装および評価が必要であり，スケーリングの観点から，複数ノードを用いた場合の高速化についてもより深く研究しなければならない．また，本研究では GPU による多倍長精度演算に CUMP を採用したが，高速化の障壁と考えられる要素がいくつか見受けられた．

- CUMP へデータ転送を行う際，受け取り先の転送位置を指定できない．これは，例えば各 GPU が計算した結果をあるベクトルへ集めるために必要で，現状は CPU 側に一旦書き戻す操作が必要になりオーバーヘッドとなる．
- データ転送に非同期 API がない．マルチ GPU による計算であるため CPU-GPU，GPU-GPU 間のメモリ転送は非同期としてオーバーラップできた方が良い．しかしながら，CUMP ではメモリレイアウトが GMP と異なるため，CPU-GPU 間のメモリ転送の際は一時領域へコピーする必要がある．
- GPU カーネル内で計算のための一時領域としてシェアードメモリを利用できない．本研究では，グローバルメモリに一時領域を確保したが，シェアードメモリを有効利用することで，計算規模を拡大できると考えられる．

多倍長精度演算の GPU 実装は，マルチ GPU で実装しなければ大規模計算および並列化，高速化を行うことは難しい．解法によっては非常に多くのメモリを使用することになり，1 台あたりのメモリが CPU に比べて少ない GPU ではマルチ GPU でなければ解くこと自体が難しくなってしまう．そのため，GPU による多倍長精度演算ライブラリは，マルチ GPU を行うためのサポートを提供することが必要となるだろう．

謝辞 本研究は筑波大学計算科学研究センターが公募している学際共同利用プログラムのプロジェクトとして採択されている．今回 HA-PACS を使用したプロジェクトとして本研究を採択していただいた学際共同利用委員会および筑波大学計算科学研究センターの皆様に深謝する．

参考文献

[1] Ikuno, S., Kawaguchi, Y., Fujita, N., Itoh, T., Nakata, S. and Watanabe, K.: Iterative Solver for Linear System Obtained by Edge Element: Variable Preconditioned Method With Mixed Precision on GPU, *IEEE Transactions on Magnetics*, Vol. 48, pp. 467–480 (2012).

[2] Commer, M., Maia, F. R. and Newman, G. A.: Iterative Krylov solution methods for geophysical electromagnetic simulations on throughput-oriented processing units, *International journal of High Performance Computing Applications*, Vol. 26, No. 4, pp. 378–385 (2012).

[3] The GNU Project: GMP, The GNU Multiple Precision Arithmetic Library, `http://gmplib.org/`.

[4] T. Nakayama and D. Takahashi: Implementation of Multiple-Precision Floating-Point Arithmetic Library for GPU Computing, *Proc. 23rd IASTED International Conference on Parallel and Ditributed Computing and Systems (PDCS 2011)*, pp. 343–349 (2011).

[5] Mian Lu, Bingsheng He and Qiong Lio: Supporting extended precision on graphics processors, *Proceedings of the Sixth International Workshop on Data Management on New Hardware (DaMoN 2010)*, No. ACM 978-1-4503-0189-3 (2010).

[6] INRIA and others: MPFR, The GNU MPFR Library, `http://www.mpfr.org/`.

[7] 幸谷智紀：PC Cluster 上における多倍長数値計算ライブラリ BNCPack の並列分散化，Linux Conference 抄録集 第 1 巻， No. CP-14 (2003).

[8] H. Igarashi: On the Property of the CurlCurl Matrix in Finite Element Analysis With Edge Elements, *IEEE Transaction on Magnetics*, Vol. 37, No. 5, pp. 3129–3132 (2001).

[9] K. Abe and S. L. Zhang: A variable preconditioning using the SOR method for GCR-like methods, *Int.J.Number.Anal Model.2*, No. 2, pp. 118–128 (2002).

[10] 筑波大学計算科学研究センター：HA-PACS プロジェクト, `http://www.ccs.tsukuba.ac.jp/CCS/research/project/ha-pacs`.

[11] Network-Based Computing Laboratory: MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE, `http://mvapich.cse.ohio-state.edu/index.shtml`.

[12] J. Dongarra: Sparse Matrix Storage Formats, `http://web.eecs.utk.edu/~dongarra/etemplates/node372.html`.

[13] A. Cevahir, A. Nu kada and S. Matsuoka: Fast Conjugate Gradients with Multiple GPUs, *Computation Science (ICCS 2009) part I*, pp. 893–903.

[14] 吉澤大樹，高橋大介：GPU における CRS 形式疎行列ベクトル積の自動チューニング，情報処理学会研究報告，Vol. 2012-HPC-135 No.31 (2012).

[15] The University of Florida: Sparse Matrix Collection, `http://www.cise.ufl.edu/research/sparse/matrices/`.