Vol.2017-HPC-160 No.14
2017/7/27

GPU クラスタ上における階層型行列計算の最適化

大島 聡史[1,a)]　山崎 市太郎[2]　伊田 明弘[3]　横田 理央[4]

概要：階層型行列は小さな密行列と低ランク近似行列から構成される行列である．密行列を階層型行列によって近似することで, 大規模な計算をより少ないメモリ量で行うことが可能となる．しかし階層型行列を用いた計算は複雑であるため, 最適化が求められている．我々はこれまで階層型行列を用いた境界要素法による静電場解析問題の実装と評価をマルチコア CPU やメニーコアプロセッサにて実施してきた．本稿では, 階層型行列を係数行列に持つ線形方程式に対する反復法を対象として, GPU クラスタ上での性能評価や最適化に取り組んだ結果を示す．主要な計算部である階層型行列ベクトル積計算を構成する密行列ベクトル積計算を MAGMA BLAS に行わせることで高速化を目指したところ, GPU カーネル起動のオーバーヘッドにより実行時間が増大したが, BATCHED MAGMA を用いることで大幅に性能が改善した．実験環境としては TSUBAME 2.5(最大 8 ノード/1 ノードあたり 1GPU) および Reedbush-H(最大 8 ノード/1 ノードあたり 1GPU) を使用し, それぞれ 8 ノードまで性能向上は得られたが, ノード数を増やした場合には MPI 処理の時間も目立ってきており, さらなる最適化が求められる結果となった.

1. はじめに

大規模な計算や複雑な計算を高速に行いたいという需要は大きく, そのために高速な演算装置や大容量の記憶装置が求められている．しかし, 計算ハードウェアのさらなる性能向上には様々な技術的困難が存在している.

2004 年頃まで, プロセッサの性能は半導体プロセスの微細化に基づき動作周波数を向上させることにより向上してきた．しかし今日では物理的な大きさの限界に起因する微細化の難しさやリーク電流の増大が問題となり, プロセッサの動作周波数の向上は困難となっている．そのため, 多数の計算コアを搭載しプロセッサ全体で高性能を達成するプロセッサの普及が進んでいる．現在 HPC 分野にて多く用いられている Intel 社のマルチコア CPU「Xeon」シリーズ (Broadwell-EP) は, 1 ソケットに最大 20 程度の計算コアを搭載し,1 コアあたり 2 スレッド同時実行が可能, 総理論演算性能は最大 1TFLOPS 程度である．さらに同社の最新のメニーコアプロセッサ「Xeon Phi」シリーズ (Knights Landing) は, 最大で 72 の計算コアを搭載し, 同時実行可能スレッド数は 288 スレッド (1 コアあたり 4 スレッド),3TFLOPS 程度の演算性能を備えている．マルチコア CPU やメニーコアプロセッサよりも並列計算による高い演算性能に特化した GPU は, さらに計算コア数や理論演算性能が高く, 最新の HPC 向け GPU「Tesla P100」(Pascal) では計算コア (CUDA コア)3584 個で 5TFLOPS 以上の性能を備えている．しかしマルチコア CPU よりもメニーコアプロセッサ, メニーコアプロセッサよりも GPU の方が単体計算コアの性能は低い傾向にあるため, 高い演算性能を十分に発揮するためには高い並列度をうまく活用できるようなアルゴリズムや実装が非常に重要である．記憶装置, 主にメインメモリについて見てみると, マルチコア CPU では現在 DDR3 や DDR4 メモリが主流であり, 転送速度も年々徐々に増加している．メニーコアプロセッサや GPU では従来から同世代の DDR 系のメモリよりも高速な GDDR 系のメモリが普及しており, さらに MCDRAM (Multi Channel DRAM) や HBM (High Bandwidth Memory) といったより高速な三次元積層メモリの採用も進んでいる.

一方, 大規模な計算を行いたいという需要は大きい．特に大規模な密行列を用いた計算については, キャッシュの活用などの手法を用いた高速計算に関する研究が多く行われている．しかしながら, 大規模な密行列を扱うには多くのメモリ容量も必要である．今日の計算機環境においては, 演算性能 (FLOPS) の向上に対してメモリ転送速度 (Byte/sec) の向上は遅れており, メモリ容量 (Byte) の増加も遅れている．特に高速な三次元積層メモリは価格や仕様の制限により搭載容量が少ない傾向にあり, 限られた容量の高速メモリをどのように活用するかは今日の HPC において重要な

1　九州大学 情報基盤研究開発センター
2　Electrical Engineering and Computer Science Department, University of Tennessee
3　東京大学 情報基盤センター
4　東京工業大学 学術国際情報センター
a)　ohshima@cc.kyushu-u.ac.jp

課題となっている．そこで近年注目されているのが, 階層型行列法 (Hierarchical matrices ($\mathcal{H}-matrices$)[1] である．階層型行列法では, 対象となる行列は多数の部分行列に分割され, その部分行列の多くは低ランク行列により近似される．階層型行列法を用いれば同じメモリ容量でもより大きな規模の密行列を扱うことが可能となるため, 大規模な計算が可能となることが期待される．その一方で階層型行列を用いた計算は通常の密行列を用いた計算と比べて複雑であるため, 新たな演算法の開発や高速化実装が強く求められている．

我々は, 静電場解析を主な対象問題として, 階層型行列を係数行列に持つ線形方程式を反復法で解くための研究を進めている．例えば [2] においては, 反復法に占める実行時間の割合が大きな行列ベクトル積 (階層型行列ベクトル積) に着目し, メニーコアプロセッサ上での実装法と性能について報告した．本稿では対象計算ハードウェアを GPU として, 階層型行列ベクトル積計算の実装と性能について報告する．さらに, より大規模な計算を行うことを目指し, 複数 GPU の活用についての検討や評価を行った結果についても報告する．

本稿の構成は以下の通りである．2 章では階層型行列の構成とそれを用いた計算について述べる．3 章では対象とする行列や計算機環境について述べ, CPU を用いた階層型行列に対する反復法の実装と性能を示す．4 章では GPU を用いた階層型行列に対する反復法の実装について検討し, 特に階層型行列ベクトル積計算や MPI 通信時間を中心とした性能評価を行う．5 章はまとめの章とする．

2. 階層型行列を用いた計算

2.1 階層型行列

本節では N 次元実正方行列 $\overline{A} \in \mathbb{R}^{N\times N}$ について考える．本論文で扱う階層型行列（A と表記する）とは，$\overline{A}$ を多数の部分行列に分割した上で，それら部分行列の大半を低ランク行列を用いて近似したものである．

ここで, N 次元正方行列の行に関する添え字の集合を $I := \{1,\cdots,N\}$, 列に関する添え字の集合を $J := \{1,\cdots,N\}$ と表す．直積集合 $I\times J$ を重なりなく分割して得られる集合の中で, 各要素 m が I と J の連続した部分集合の直積であるものを M とする．すなわち任意の $m \in M$ は $s_m \subseteq I, t_m \subseteq J$ を用いて $m = s_m \times t_m$ と表される．ある m に対応する $\overline{A}$ の部分行列を

$$A|^m_{s_m\times t_m} \in \mathbb{R}^{\#s_m\times\#t_m} \tag{1}$$

と書く．ここで # は集合の要素数を与える演算子である．階層型行列では, 大半の m について $A|^m_{s_m\times t_m}$ の代わりに以下の低ランク表現 $\tilde{A}|^m$ を用いる．

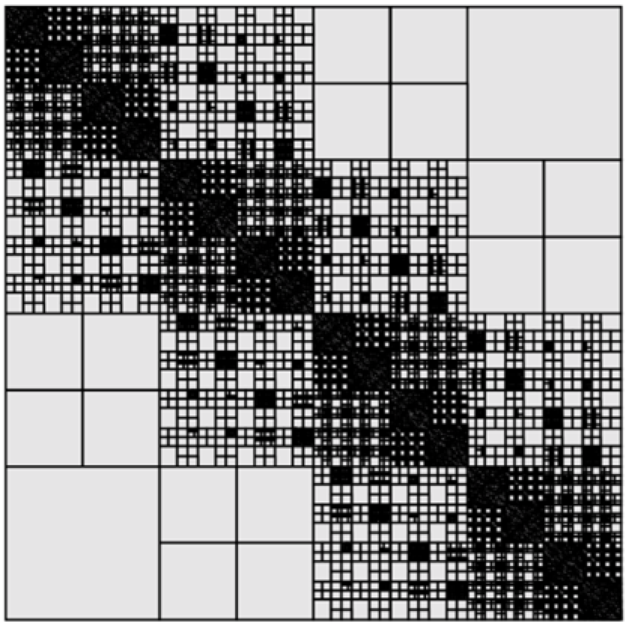

図 1　階層型行列の例

$$\begin{aligned} &\tilde{A}|^m := V_m\cdot W_m\\ &V_m \in \mathbb{R}^{\#s_m\times r_m}\\ &W_m \in \mathbb{R}^{r_m\times\#t_m}\\ &r_m \le min(\#s_m,\#t_m) \end{aligned} \tag{2}$$

ここで $r_m \in \mathbb{N}$ は行列 $\tilde{A}|^m$ のランクである．すなわち, 低ランク行列 $\tilde{A}|^m$ とは, 密行列 $A|^m_{s_m\times t_m}$ を V_m と W_m の積により近似した行列である．**図 1** に階層型行列の一例を示す．図 1 の濃く塗りつぶされた部分行列が $A|^m_{s_m\times t_m}$ に, 薄く塗りつぶされた部分行列が $\tilde{A}|^m$ に対応する．

本稿では階層型行列に関する演算の中でも行列ベクトル積 (階層型行列ベクトル積計算) を中心に論じる．ある階層型行列 A に関するデータ量 $N(M)$ は，m に対応する部分行列に関するデータ量 $N(m)$ を用いて以下のように表される．

$$N(M) = \sum_{m\in M} N(m) \tag{3}$$

$$N(m) := \begin{cases} \#s_m\times\#t_m & \text{m が密行列の場合}\\ r_m\times(\#s_m+\#t_m) & \text{m が低ランク行列の場合}\end{cases} \tag{4}$$

r_m が $\#s_m$ や $\#t_m$ と比べて十分に小さい場合，$r_m\times(\#s_m+\#t_m)$ は $\#s_m\times\#t_m$ と比べて小さい値となり, 低ランク行列による表現はデータ量の点で有利となる．その結果, 密行列を用いる場合と比べ，行列ベクトル積等の行列演算に必要な演算量や行列の保持に必要なメモリ量を低減することができる．

2.2 階層型行列ベクトル積

階層型行列は密行列を近似したものであることから, 階層型行列に対して反復法を適用するには, 密行列に対する計算と同様の計算を階層型行列に対しても行う必要がある. 本節では, 以下のような階層型行列ベクトル積について考える.

$$\begin{aligned} Ax &\to y \\ x, y &\in \mathbb{R}^N \end{aligned} \tag{5}$$

本論文ではこの演算の実施手順として，各部分行列毎に行列積を実行し，その結果を統合することで最終的な結果 y を得るという，最も自然でかつ効率的と考えられる方法を採用する．密行列により表現されている部分行列 $A|^m_{s_m \times t_m}$ については

$$A|^m_{s_m \times t_m} \cdot x|_{t_m} \to \hat{y}|_{s_m} \tag{6}$$

を計算する. ここで,$x|_{t_m}$ は x の各要素のうち tm に対応する要素のみを抜き出して生成した $\#tm$ 個の要素からなるベクトルである. $\hat{y}|_{s_m}$ は $\#s_m$ 個の要素を持つベクトルであり, 各要素は y の s_m に対応する要素の部分積の一つとなる.

次に, 低ランク表現が用いられている行列 $\tilde{A}|^m$ に関しては,$c \in \mathbb{R}^{r_m}$ として, まず

$$Wm \cdot x|_{t_m} \to c|_{r_m} \tag{7}$$

を計算し, さらに

$$Vm \cdot c|_{r_m} \to \hat{y}|_{s_m} \tag{8}$$

を計算することで, $\tilde{A}|^m \cdot x|_{t_m} = Vm \cdot Wm \cdot x|_{t_m} \to \hat{y}|_{s_m}$ が得られる.

それぞれの部分行列について $\hat{y}|_{s_m}$ を計算した後,

$$\sum_{m \in M} \hat{y}|_{s_m} \to y \tag{9}$$

のようにこれらを統合すれば, 最終的な階層型行列ベクトル積の結果を得ることができる.

2.3 対象とするプログラムと並列化手法

本稿では階層型行列に関する計算の実装として ppOpen-APPL/BEM[3] に含まれる HACApK 1.0.0 を元に修正を加えたプログラムを用いる. ppOpen-APPL/BEM は, JST CREST「自動チューニング機構を有するアプリケーション開発・実行環境: ppOpen-HPC」[4] の構成要素の 1 つであり, 境界要素法 (Boundary Element Method, BEM) の実装において HACApK[5] を用いた階層型行列による計算を行っている. この中に含まれる階層型行列を係数行列に持つ連立一次方程式を BiCGSTAB 法を用いて解くコードを使用する. ただし, 低ランク近似アルゴリズムについてはライブラリに含まれている ACA 法を使用せず, 新たに実装した ACA+法 [6] を用いている. GPU 上で行われる計算については, 元々 Fortran を用いて書かれていたプログラムを C 言語に変更したものを用いる. この理由は, 後述するように, 利用するライブラリの都合によるものである.

対象プログラムは MPI による並列化と OpenMP による並列化の両方に対応している. 並列化手法の詳細は説明は文献 [5] に記載されているが, 特に以下のような特徴がある. プロセスやスレッド毎の負荷をある程度均一にするためのアルゴリズムが含まれてはいるものの, ある程度の負荷不均衡が発生することは避けにくい. また割り当ての結果, 元の係数行列の一行に影響を与える低ランク行列や小密行列は複数のプロセスやスレッドにまたがって配置される可能性が極めて高い.

3. 問題設定

3.1 評価環境

本稿では**表 3.1** に示す 2 種類のスーパーコンピュータシステムを用いて性能評価を行う.

第 1 のシステムは東京工業大学 学術国際情報センターに設置されている TSUBAME 2.5[7] である. CPU として Xeon X5670,GPU として Tesla K20X を搭載しており, プロセッサの世代は新しくないが動作実績の蓄積されたシステムである. 第 2 のシステムは東京大学 情報基盤センターに設置されている Reedbush (Reedbush-H)[8] である. Xeon E5-2659 v4 CPU および Tesla P100 GPU はともに HPC 用途に大規模に供給されているプロセッサとしては最新世代のプロセッサである.

3.2 対象とする行列

本稿では**表 3.2** に示す階層型行列を対象問題とする. この行列は境界要素法を用いた静電場解析においてあらわれる行列である. 表中のリーフ数とは低ランク行列による近似行列の組数および小密行列数の合計数である. 階層型行列における近似行列または小密行列はツリーにおける葉 (リーフ) に相当するため, 本稿ではリーフという呼称を用いる. スレッド並列化を行った場合は各スレッドにリーフが割り当てられるが, 2.3 節にて述べたように, その割り当て数量は均等とは限らない.

3.3 CPU による性能

本題である GPU を用いた際の性能に対する比較対象として, CPU のみで対象問題を実行した際の性能を測定する.

図 2 は対象プログラムの主要な計算部分である BiCGSTAB 法の反復計算部 (OpenMP 版) の実コードである. 計算アルゴリズムは一般的な BiCGSTAB 法そのものである. 反復計算部全体が OpenMP の parallel 指示文によって囲まれており, 本コード中に計算処理そのもの

表 1　実験環境

		TSUBAME 2.5	Reedbush-H
CPU	型番	Xeon X5670 (Westmere-EP)	Xeon E5-2695 v4 (Broadwell-EP)
	ノードあたり搭載数	2	2
	動作周波数	2.93 GHz - 3.196 GHz	2.1 GHz - 3.3 GHz
	コア数	6 / socket	18 / socket
	理論演算性能	70.4 GFLOPS / socket	604.8 GFLOPS / socket
	メインメモリ容量	54 GB / node	256 GB / node
	メモリ転送速度	32 GB/s / socket	76.8 GB/s / socket
GPU	型番	Tesla K20X (Kepler)	Tesla P100 (Pascal)
	ノードあたり搭載数	3	2
	1GPU あたりメモリ容量	6 GB	16 GB
	1GPU あたり理論演算性能 (DP)	1.31 TFLOPS	4.8 - 5.3 TFLOPS
	1GPU あたりメモリ転送速度	250 GB/s	732 GB/s
	ホストとの接続	PCI-Express Gen.2 x16 (8 GB/sec)	PCI-Express Gen.3 x16 (16 GB/sec)
	GPU 間の接続	PCI-Express Gen.2 x16 (8 GB/sec)	NVLink 2 (20 GB/sec x2)
ノード間接続		InfiniBand QDR 4X x2 ((32Gbps x2)/node)	InifiniBand EDR 4X x2 ((56Gbps x2)/node)

表 2　対象とする階層型行列の構成

行列名	100ts
行数	101,250
総リーフ数	222,274
近似行列数	89,534
小密行列数	132,740
H 行列容量	2,050 MByte

が書かれている部分の多くには OpenMP の workshare 指示文などが適用されていることから，反復計算部の多くの部分が並列計算されることがわかる．また一部の演算は別の関数にて実装されている．図中で実行されている HACApK_adot_body_lfmtx_hyp 関数が階層型行列ベクトル積と MPI 通信を行う関数であり，**図 3** が階層型行列ベクトル積の実体である．ここでは各スレッドがスレッド ID を元に計算範囲を定めたうえで，近似行列ベクトル積 (図中 A 部) や小密行列ベクトル積 (図中 B 部) を行い，結果の足しあわせ (図中 C 部) を行っている．各リーフにおける階層型行列ベクトル積計算を各 CPU コアが逐次計算している点は本実装の特徴の一つであると言える．もちろん，この階層型行列ベクトル積計算を構成する近似行列ベクトル積や小密行列積にも並列性はあるため並列化が可能ではあるが，これらの計算には SIMD 化も有効であることや全体として多数のリーフが存在することから，現在のようなプログラム構造が採用されている．

BiCGSTAB 法全体および階層型行列ベクトル積計算と MPI 通信時間の実行時間を測定した．はじめに各計算機環境にて 1 ノードのみを使用して計算を実行した．OpenMP 並列化を有効化し，スレッド数は実行環境に搭載された

```
!$omp parallel
反復計算前の変数初期化等の一部処理は省略
 do in=1,mstep
   if(zrnorm/bnorm<eps) exit
!$omp workshare
   zp(:nd) =zr(:nd)+beta*(zp(:nd)-zeta*zakp(:nd))
   zkp(:nd)=zp(:nd)
!$omp end workshare
   call HACApK_adot_lfmtx_hyp(zakp,st_leafmtxp,st_ctl,zkp,wws,wwr,isct,irct,nd)
!$omp barrier
!$omp single
   znom=HACApK_dotp_d(nd,zshdw,zr); zden=HACApK_dotp_d(nd,zshdw,zakp);
   alpha=znom/zden; znomold=znom;
!$omp end single
!$omp workshare
   zt(:nd)=zr(:nd)-alpha*zakp(:nd)
   zkt(:nd)=zt(:nd)
!$omp end workshare
   call HACApK_adot_lfmtx_hyp(zakt,st_leafmtxp,st_ctl,zkt,wws,wwr,isct,irct,nd)
!$omp barrier
!$omp single
   znom=HACApK_dotp_d(nd,zakt,zt); zden=HACApK_dotp_d(nd,zakt,zakt);
   zeta=znom/zden;
!$omp end single
!$omp workshare
   u(:nd)=u(:nd)+alpha*zkp(:nd)+zeta*zkt(:nd)
   zr(:nd)=zt(:nd)-zeta*zakt(:nd)
!$omp end workshare
!$omp single
   beta=alpha/zeta*HACApK_dotp_d(nd,zshdw,zr)/znomold;
   zrnorm=HACApK_dotp_d(nd,zr,zr); zrnorm=dsqrt(zrnorm)
   nstp=in
   call MPI_Barrier( icomm, ierr )
   en_measure_time=MPI_Wtime()
   time = en_measure_time - st_measure_time
   if(st_ctl%param(1)>0 .and. mpinr==0) print*,in,time,log10(zrnorm/bnorm)
!$omp end single
 enddo
!$omp end parallel
```

図 2　BiCGSTAB 法の反復計算部の実コード

CPU 1 ソケット上の物理コア数までの数パターンを試した．さらに，複数ノードを用いた MPI + OpenMP ハイブリッド並列化による性能についても測定した．いずれの実行環境も 1CPU に 2 ソケットの CPU を搭載しているが，今回は最も単純な問題設定の 1 つであると思われる，1 ノードあたり 1MPI プロセスを起動し各ノードでは 1CPU ソケットのみを使用する，という設定にて実行時間を測定した．

TSUBAME 2.5 ではコンパイラとして Intel icc/ifort

```
ith = omp_get_thread_num()
ith1 = ith+1
nths=ltmp(ith); nthe=ltmp(ith1)-1
allocate(zaut(nd)); zaut(:)=0.0d0
allocate(zbut(ktmax))
ls=nd; le=1
do ip=nths,nthe
  ndl   =st_leafmtxp%st_lf(ip)%ndl   ; ndt   =st_leafmtxp%st_lf(ip)%ndt   ; ns=ndl*ndt
  nstrtl=st_leafmtxp%st_lf(ip)%nstrtl; nstrtt=st_leafmtxp%st_lf(ip)%nstrtt
  if(nstrtl<ls) ls=nstrtl; if(nstrtl+ndl-1>le) le=nstrtl+ndl-1
  if(st_leafmtxp%st_lf(ip)%ltmtx==1)then
    kt=st_leafmtxp%st_lf(ip)%kt
    zbut(1:kt)=0.0d0
    do il=1,kt
      do it=1,ndt; itt=it+nstrtt-1
        zbut(il)=zbut(il)+st_leafmtxp%st_lf(ip)%a1(it,il)*zu(itt)
      enddo
    enddo                                                              A
    do il=1,kt
      do it=1,ndl; ill=it+nstrtl-1
        zaut(ill)=zaut(ill)+st_leafmtxp%st_lf(ip)%a2(it,il)*zbut(il)
      enddo
    enddo
  elseif(st_leafmtxp%st_lf(ip)%ltmtx==2)then
    do il=1,ndl; ill=il+nstrtl-1
      do it=1,ndt; itt=it+nstrtt-1
        zaut(ill)=zaut(ill)+st_leafmtxp%st_lf(ip)%a1(it,il)*zu(itt)     B
      enddo
    enddo
  endif
enddo
deallocate(zbut)

do il=ls,le
!$omp atomic
  zau(il)=zau(il)+zaut(il)                                             C
enddo
```

図 3　行列ベクトル積部の実コード
(HACApK_adot_body_lfmtx_hyp 関数の抜粋)

16.0.3, MPI として Intel MPI Library 5.1.3 を用いた．主なコンパイルオプションは`-qopenmp -O3 -ip -xSSE4.1 -mcmodel=large` である．一方 Reedbush-H ではコンパイラとして Intel icc/ifort 17.0.2, MPI として OpenMPI 2.0.2 を用いた．主なコンパイルオプションは`-qopenmp -O3 -ip -xCORE-AVX2 -mcmodel=large` である．いずれの実行環境においても実行時には `numactl --cpunodebind=0` を指定している．

図 4, **図 5**, **図 6**, **図 7** に実行時間の内訳を示す．いずれも縦軸 (左) は BiCGSTAB 法の 1 反復計算あたりの実行時間をあらわしているが, 階層型行列ベクトル積は 1 反復計算あたり 2 回実行されており, 内訳には 2 回分の実行時間の和が示されている．また MPI に要する時間の中でも主要な送受信時間のみが MPI Send/Recv に含まれており, プロセス間の同期時間はその他に含まれている．

1 ノードのみを用いてスレッド数を変化させて実行時間を調査した結果 (図 4, 図 5) からは, 実行時間のほぼ全てが階層型行列ベクトル積に費やされていることがわかる．TSUBAME 2.5 および Reedbush-H ともに使用スレッド数を増やすほど高い性能が得られており, スレッド数分のリニアな性能向上とはいかないものの, 最大物理コア数分まで性能が向上していることが確認できる．Reedbush-H は TSUBAME 2.5 と比べて 3.5 倍程度高速であった．各 CPU の演算性能は約 8.6 倍, メモリ転送性能は約 2.4 倍の差があり, メモリバンド幅の影響が大きいと言える．

一方, 複数ノードを用いた場合の実行結果 (図 6, 図 7) を見てみると, Reedbush-H では 8 ノードまで使用ノード数が多いほど実行時間が短くなっており, 8 ノード実行でも実行時間の半分程度が階層型行列ベクトル積計算に費やされている．しかし TSUBAME 2.5 では同じ 8 ノード実行において階層型行列ベクトル積計算/MPI/その他の処理がそれぞれ 3 割程度の時間を費やしており, 階層型行列ベクトル積計算だけを比較すれば 1 ノードよりも 8 ノードの方が時間が短いものの, 1 反復あたり実行時間は長くなってしまっている．この結果からは, 特に TSUBAME 2.5 については CPU 向けの実装や実行方法についてまだ改善の余地が残されている可能性がある．

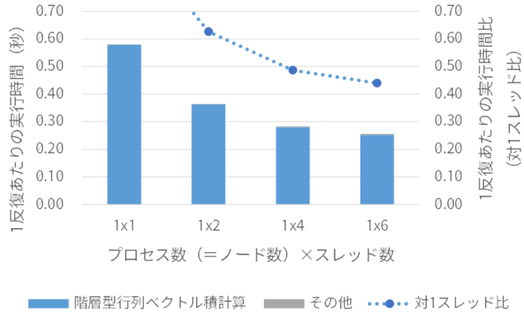

図 4　CPU による実行時間
(TSUBAME 2.5, 1 ノード)

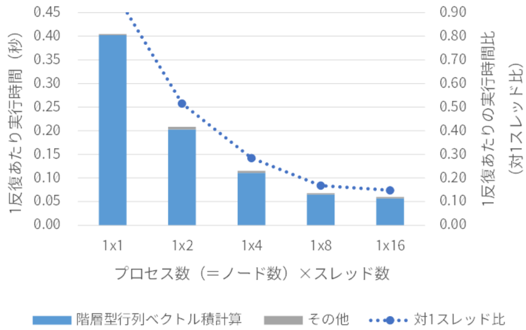

図 5　CPU による実行時間
(Reedbush-H, 1 ノード)

4.　GPU を用いた階層型行列ベクトル積計算

4.1　実装の方針

前章で確認したように,BiCGSTAB 法において特に計算時間が長いのは階層型行列とベクトルの積を求める計算 (階層型行列ベクトル積) である．そこで本節では, 階層型行列ベクトル積計算を 1GPU で高速に実行することを考える．

既に述べたように, 階層型行列ベクトル積は多数の低ランク近似行列ベクトル積および小密行列ベクトル積から構成されているため, これらの計算をいかにして GPU 上で高速に実行するかが重要である．ここで, 低ランク近似行列ベクトル積が密行列ベクトル積の組み合わせによって構成されていることから, 階層型行列ベクトル積は高速な密行列ベクトル積の実装があれば高速に行えることがわかる．密行列ベクトル積計算は BLAS (Basic Linear Alrebra

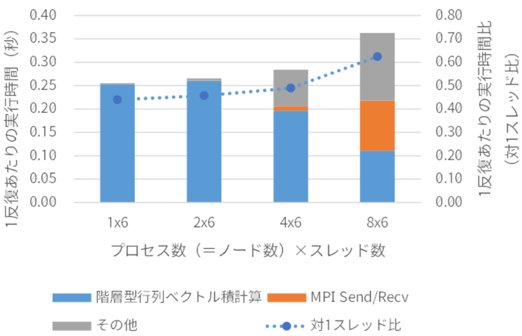

図 6　CPU による実行時間
(TSUBAME 2.5, 複数ノード)

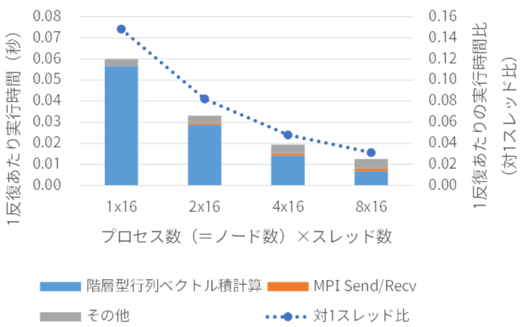

図 7　CPU による実行時間
(Reedbush-H, 複数ノード)

Subprograms)[9] 互換ライブラリにて GEMV として提供されている計算であり, 本稿で用いている GPU 向けにはすでに CUBLAS[10] などの高速な BLAS 実装が提供されている．これらを活用すれば容易に階層型行列ベクトル積の高速化ができると期待される．

しかしながら, 階層型行列ベクトル積のアルゴリズムに含まれる GEMV を GPU に対応した BLAS ライブラリに単純に置き換えた場合, 性能面で問題が生じる可能性が考えられる．第一に,BLAS ライブラリによる高速化の恩恵が特に大きいのは大規模な行列に対して計算を行う場合であるため, BLAS の実装によっては小規模な行列について十分な最適化が行われていない可能性がある．階層型行列ベクトル積においては小規模な GEMV も多数行われる場合があり, 十分な性能が得られない可能性がある．これは CPU を用いた階層型行列ベクトル積の実装に CPU 向けの BLAS ライブラリを用いた場合でも同様に生じる問題であるが, GPU を用いた実装ではさらに GPU カーネル起動のオーバーヘッドが問題となる．本稿で用いている GPU(NVIDIA 社 Tesla GPU) 向けの高性能なプログラムを作成するには主に GPU 向けプログラミング環境である CUDA[11] が用いられるが, これによって GPU に計算を行わせる単位は関数であり,GPU カーネルと呼ばれている．GPU カーネルを起動し GPU に計算を行わせる際には, CPU における関数実行と比べて非常に長い GPU カーネル起動時間が必要である．そのため, 今回のように GPU カーネルの起動回数が多い場合は大幅な実行時間の増加を引き起こしてしまう．これは OpenCL や OpenACC といった異なる GPU 向けプログラミング環境を用いた場合や, GPU 向けのライブラリを用いた場合でも同様に生じる問題である．

そこで,GPU に対応した BLAS の 1 つである MAGMA BLAS[12] に備えられた “BATCHED” 機能 (BATCHED MAGMA)[13] を用いる．BATCHED MAGMA は GEMV などの計算を 1GPU カーネル内で複数個続けて行う (“バッチ” 処理する) 機能を提供している．そのため今回のように小規模な計算を何度も実行する場合には, GPU カーネル起動のオーバーヘッドが削減され, 性能改善が行える可能性がある．

4.2　1GPU 環境における性能評価

階層型行列ベクトル積計算に含まれる GEMV 計算を MAGMA BLAS に置き換えて性能を測定する．測定においては, 単純に MAGMA BLAS に置き換えた場合の性能と, BATCHED MAGMA を用いて実装した場合の性能をそれぞれ測定し, その差を比較する．BATCHED MAGMA については, 1 反復計算ごとに CPU から GPU へ送って更新せねばならないデータの転送時間, バッチ情報を MAGMA ライブラリに登録するための時間, GPU 上で計算が行われる時間をそれぞれ測定した．なお MAGMA BLAS は BATCHED MAGMA に対する Fortran インターフェイスのサポートが確認できていなかったため, 元プログラムを Fortran から C 言語に移植し,C 言語部分から MAGMA BLAS を呼び出す構造とした．現時点では特に実行時間の長い階層型行列ベクトル積の最適化に注力しているため, それ以外の処理は CPU 上で逐次実行している．また, 複数ノード間のデータ転送については保守的な実装となっており, MPI 通信時のメモリコピーを減らす GPU Direct の活用などは適用できていない．

実行環境は前章と同様である． GPU 実行環境としては,TSUBAME 2.5 では CUDA 7.5,Reedbush-H では CUDA 8.0 を使用した． MAGMA BLAS のバージョンは 2.2 である． MAGMA BLAS の構築時には,TSUBAME 2.5 では Kepler アーキテクチャ向け (GPU_TARGET=Kepler), Reedbush-H では Pascal アーキテクチャ向け (GPU_TARGET=Pascal) の最適化オプションを適用した．

1 ノード 1GPU を用いた場合の実行時間測定結果を図 **8** に示す．バッチを使用せずに単純に GEMV 関数を用いた場合 (MAGMA) とバッチを使用した場合 (BATCHED MAGMA) とでは大きな性能差が生じていることがわかる．バッチを使わない場合には GPU カーネルをリーフ数分だけ実行する必要があり (正確には低ランク近似行列ベ

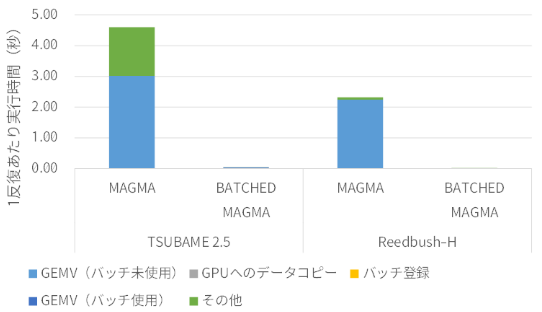

図 8 1GPU による実行時間

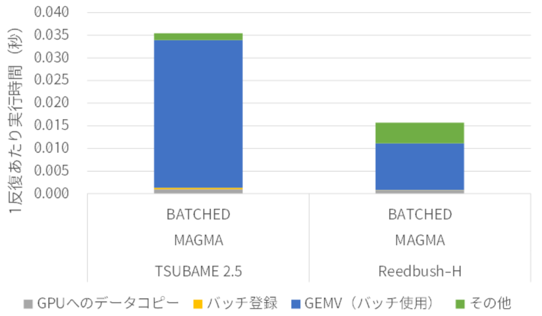

図 9 1GPU による実行時間
(BATCHED MAGMA 版のみ)

クトル積を実行するために 2 回の GPU カーネルを実行するためさらに多い) GPU カーネル実行のオーバーヘッドが大きい一方, バッチを使うことでそのオーバーヘッドが削減され短時間で計算できることが良くわかる結果となった. 図 8 から BATCHED MAGMA のみを抽出したものを**図 9** に示す. TSUBAME 2.5,Reedbush-H ともに実行時間の多くが BLAS による計算によって占められていることがわかる. 図 9 における TSUBAME 2.5 と Reedbush-H の GEMV 計算時間を比較すると Reedbush-H のほうが 3.17 倍高速であった. 各 GPU の性能差は演算性能で約 4 倍, メモリ転送性能で約 3 倍あり, 妥当な性能差であると考えられる.

4.3 複数 GPU 環境向けの実装と性能

前章にて述べたように ppOpen-APPL/BEM は MPI と OpenMP を用いた階層的な並列化に対応しており, MPI による並列化においては階層型行列全体をある程度均等にプロセスへ分割し, OpenMP による並列化では各プロセスへ割り当てられたリーフをさらにスレッドへと分割している. 前節の 1GPU を用いた実行においては,1 プロセスが存在する全てのリーフを MAGMA ライブラリを用いて計算していたが, 各 MPI プロセスが割り当てられたリーフを MAGMA ライブラリを用いて計算するという構造に容易に拡張することができる. そこで本節では, 前節での実験を複数 MPI プロセスに変更して性能評価を行った上で, さらなる性能向上の余地について検討する.

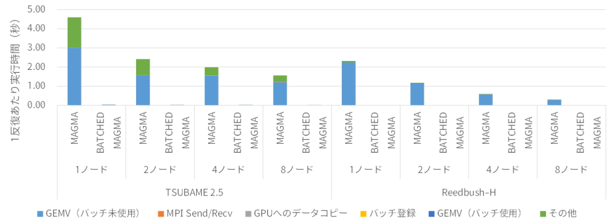

図 10 複数 GPU による実行時間

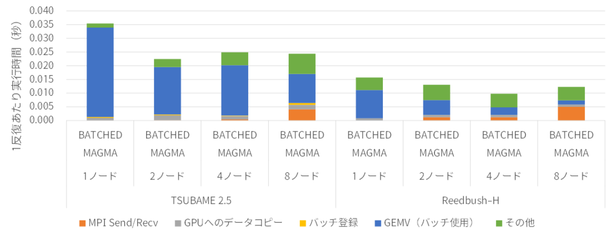

図 11 複数 GPU による実行時間 (BATCHED MAGMA 版のみ)

TSUBAME 2.5 および Reedbush-H にて複数の GPU を用いた場合の実行時間を**図 10** および**図 11** に示す. CPU のみの実行時間を測定した際と同様に, いずれの実験環境においても 1 ノードあたり 1GPU のみを利用している. 図 10 はバッチを利用していない MAGMA と利用している BATCHED MAGMA の実行時間を同時に示したものであるが, その性能差が非常に大きいため, 図 11 に BATCHED MAGMA の結果のみを改めて示している.

図 10 および図 11 に示した結果から, TSUBAME 2.5 と Reedbush-H ともに, 階層型行列ベクトル積の実行時間は 8 ノード (8GPU) まで短くなっている一方, MPI による通信の時間も目立ってきていることがわかる. 特に Reedbush-H では 8GPU 実行時において MPI 処理の実行時間割合が計算時間の割合を大きく越えてしまっている. ただし本項の実装では GPU Direct の利用など MPI 通信の最適化にまだ改善の余地がある. また TSUBAME 2.5 も Reedbush-H もノード数を増やすとその他として示されている部分の割合が大きくなっているものの, 階層型行列ベクトル積以外の計算についても GPU 化を行うことである程度の改善が行えると考えられる.

最後に各実行環境における 1CPU (スレッド数=物理コア数),1CPU×8 ノード, 1GPU (MAGMA および BATCHED MAGMA), 1GPU×8 ノードを用いた際の実行時間を比較したグラフを**図 12** に示す. 実行時間の差が大きいため対数グラフであることに注意されたい. TSUBAME 2.5 も Reedbush-H も同様に, バッチ機能を使わない MAGMA による実行時間は CPU と比べても非常に低速であるが, バッチ機能を使うことで大きく性能向上していることが確認できる. また TSUBAME 2.5 の 8CPU や Reedbush-H の 8GPU では MPI 通信時間の割合も目立っており, 通信の最適化も必要であることが伺える.

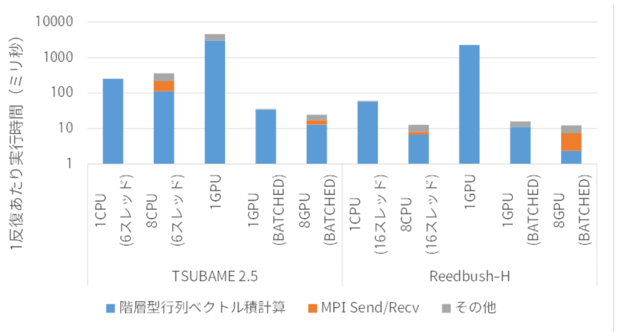

図 12 1CPU, 8CPU, 1GPU, 8GPU による実行時間

5. おわりに

本稿では静電場解析問題にて生じる係数行列を対象とした反復法の高速化に向けて, GPU を用いた階層型行列に対する計算, 特に階層型行列ベクトル積の実装と性能評価を行った．階層型行列ベクトル積の実装には密行列ベクトル積計算 (GEMV) を提供するライブラリが活用可能であるが, その回数が多いため, 単純に GEMV 計算を置き換えると GPU カーネル起動のオーバーヘッドにより大幅に性能低下することが確認された．しかし BATCHED MAGMA によりオーバーヘッドを削減することで性能が改善し, TSUBAME 2.5 では 1CPU ソケット (6 コア) と比べて階層型行列ベクトル積計算の実行時間を 13.4%まで削減, Reedbush-H では同様に 19.6%まで削減することができた．さらに複数ノードを用いることで, TSUBAME 2.5 では 8GPU で 5.11%まで,Reedbush-H では 8 ノードで 4.16%まで階層型行列ベクトル積計算の実行時間を削減することができた．一方 8MPI の時点で MPI による通信時間の割合が無視できない大きさになっており, 通信の最適化も重要であることがわかる結果となった．

以上のように, 複数 GPU 環境においても対象プログラムをある程度性能向上させることができた．一方, 階層型行列の大きさや形状が異なる場合の性能評価, GPU Direct を用いるなどして MPI 通信性能を向上させること, ノード内に複数 GPU がある場合に向けた最適化, GEMV 関数自体の改善, などの点についてはさらに検討や実装の余地があり, 引き続き取り組んでいく予定である．

謝辞 本研究は JSPS 科研費 17H01749, 科学技術振興機構戦略的創造研究推進事業 (JST/CREST), German Priority Programme 1648 Software for Exascale Computing (SPPEXA-II), 大規模学際情報基盤共同利用・共同研究拠点 (JHPCN) の支援を受けています．

参考文献

[1] Börm, S., Grasedyck, L., Hackbusch, W., “Hierarchical matrices”, Lecture note No. 21 of the Max Planck Institute for Mathematics in the Sciences (2003).
[2] 大島聡史, 伊田明弘, 河合直聡, 塙敏博, “階層型行列ベクトル積のメニーコア向け最適化”, 情報処理学会 研究報告 (HPC-155), Vol.2016-HPC-155, pp.19 (2016).
[3] Takeshi Iwashita, Akihiro Ida, Takeshi Mifune, Yasuhito Takahashi, “Software Framework for Parallel BEM Analyses with H-matrices Using MPI and OpenMP”, Procedia Computer Science 108C, pp.22002209 (2017).
[4] ppOpen-HPC — Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT), `http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/` (accessed 2017-06-28).
[5] Akihiro Ida, Takeshi Iwashita, Takeshi Mifune, Yasuhito Takahashi, “Parallel Hierarchical Matrices with Adaptive Cross Approximation on Symmetric Multiprocessing Clusters”, Journal of Information Processing, Vol.22, No.4, pp.642-650 (2014).
[6] Börm S., Grasedyck L. and Hackbusch W., “Hierarchical Matrices”, Lecture Note, Max-Planck-Institut fur Mathematik (2006).
[7] TSUBAME 計算サービス, `http://tsubame.gsic.titech.ac.jp/` (accessed 2017-06-28).
[8] Reedbush スーパーコンピュータシステム [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/system/reedbush/` (accessed 2017-06-28).
[9] BLAS (Basic Linear Algebra Subprograms), `http://www.netlib.org/blas/` (accessed 2017-06-28).
[10] cuBLAS :: CUDA Toolkit Documentation, `http://docs.nvidia.com/cuda/cublas/index.html` (accessed 2017-06-28).
[11] CUDA Zone — NVIDIA Developer, `https://developer.nvidia.com/cuda-zone` (accessed 2017-06-28).
[12] MAGMA Blas Webpage, `http://icl.cs.utk.edu/magma/index.html` (accessed 2017-06-28).
[13] Tingxing Dong, Azzam Haidar, Stanimire Tomov, Jack Dongarra, “Optimizing the SVD Bidiagonalization Process for a Batch of Small Matrices”, Procedia Computer Science, Vol.108, pp.1008-1018 (2017).