Vol.2016-HPC-155 No.43
2016/8/10

# 一般化菱形行列冪カーネルのための領域分割アルゴリズム

須田 礼仁[1]

**概要**：ステンシル計算の時空間タイリングは多くの手法が知られているが，それを疎行列ベクトル積に一般化した行列冪カーネルについては手法の開発が遅れている．従来手法は通信が 1 回で済むが計算の重複が多いという欠点があった．我々は計算の重複がない手法を提案する．これはステンシル計算で菱形方式として知られているものに対応する．小規模な行列において性能評価も行う．

## 1. はじめに

京コンピュータで 10 Pflops を超えたスーパーコンピュータであるが，最近中国の神威は 1000 万コアを用いて 93 Pflops を達成し，1 Eflops への道は続いている．複数のプロセッサを協調させて大規模計算を行うためにはプロセッサ間通信が必要であり，通信の所要時間は並列性の増大に伴い漸増することが予想されている．またコアとメモリの間のデータ移動も一種の通信と考えると，深さを増すメモリ階層によるデータ移動の所要時間も通信時間とみなしうる．GPU 等のアクセラレータを用いた計算においても，CPU と GPU との間の通信，ならびに GPU 内部でのメモリ階層間でのデータ移動が性能に大きな影響を与える．このような背景のもと，通信のコストを削減するようなアルゴリズムの研究が必要と考えられている．このようなアルゴリズムは**通信削減アルゴリズム** (Communication-Avoiding Algorithms: CA Algorithm) と呼ばれている．たとえば QR 分解の通信削減アルゴリズムである TSQR [1] はその高い有効性がよく知られている．

著者らのグループでも通信削減アルゴリズムの研究を進めてきた．なかでも大規模な連立一次方程式や固有値問題を解くために広く用いられる Krylov 部分空間法について通信削減型のアルゴリズムを研究してきている [5], [6], [7], [9], [11]．本稿では，その中核的計算のひとつである行列冪カーネル（Matrix Powers Kernel: MPK）を取り上げる．

行列冪カーネルはステンシル計算で知られている時空間タイリングを一般の疎行列に拡張したものと考えることができる．ステンシル計算では非常にたくさんの時空間タイリングの手法が知られているが，それらを一般の疎行列に拡張することはこれまであまり行われて来ていない．時空間タイリングではそれらを定義するために空間座標軸が重要な役割を果たしているが，一般の疎行列ではそれに該当するものがないことが理由のひとつと考えられる．時空間タイリングを一般疎行列に拡張するには，空間座標に代わるような概念を導入することが必要である．

一般の疎行列の行列冪カーネルの手法としてよく知られているのは，Hoemmen の論文 [4] に含まれる PA1, PA2 という手法である．実際に適用した事例として [8] などがある．これらの手法では通信の回数は削減されるが，計算の重複が多量に発生してしまうという課題があった．これに対し著者は昨年，計算の重複なしに計算する行列冪カーネルの手法を報告した [11]．ただし著者がこのときに論文として発表できたのは 2 ページの概要のみであり，アルゴリズムの詳細を全く説明することができていない．本稿では，その具体的な手法を説明する．また，小規模な行列を用いた予備実験の結果を示す．

## 2. 行列冪カーネルと時空間タイリング

### 2.1 行列冪カーネル

行列冪カーネルとは，正方形の疎行列 $A$ とベクトル $x$ に対し，$x^{(k)} = A^k x$ $(k = 1, 2, \ldots, K)$ を計算するカーネルである．ここで $K$ は正整数である．実際に計算したい行列冪カーネルには 2 種類がある．

(1) すべての $k$ に対する結果を保持する．

$$(x^{(1)}, x^{(2)}, \ldots, x^{(K)}) = (Ax, A^2x, \ldots, A^Kx)$$

(2) 最後の $x^{(K)}$ のみ保持する．途中の $x^{(k)}$ $(k = 1, 2, \ldots, K-1)$ は保持しなくてよい．

通信削減アルゴリズムでは，これらの行列冪カーネルを少ない通信回数で計算したい．

なお，以下では前提として，$A^2$ などの行列冪は陽には計

[1] 東京大学
The University of Tokyo, Bunkyo-ku, Tokyo 113-8656, Japan

算しないものとする．$A^2$ は $A$ に比べて非零要素数が非常に多くなりうるため，多くの場合 $x^{(2)} = (A^2)x$ と計算するよりも $x^2 = A(Ax)$ と計算したほうが高速であるからである．このとき，上記の 2 種類の計算は途中結果を保存するかどうかだけが違いとなるので，いま注目している通信回数に関しては違いがなくなる．しかし，メモリ確保や計算強度などは異なるので，シングルプロセッサでのチューニングでは差が出る．

また，本稿では最も簡単な場合として，疎行列は対称な非零パターンを持ち，対角要素は非零であるとする．このとき，疎行列は無向グラフと対応づけられる．すなわち，行や列の各添え字がグラフの頂点に対応し，非零要素がグラフの辺に対応する．著者らは通信削減 CG 法の研究をしており，CG 法の場合は対称正定値行列が対象なので，この条件を満たすことが多い．

### 2.2 時空間タイリング

行列冪カーネルはステンシル計算の時空間タイリングに対応する．ステンシル計算の時空間タイリングには多数の研究成果があるが，本稿の主目的ではないので，以下では最低限の説明にとどめる．

もっとも説明が行いやすい事例は 1 次元の熱伝導方程式

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

などである．初期条件と Dirichlet 境界条件が設定されているとする．これを，空間は 2 次精度中心差分，時間は陽的 Euler 法で離散化すると

$$u[i, k+1] = (1-2r)u[i,k] + r(u[i+1,k] + u[i-1,k])$$

となる．ここで $i$ は空間方向の格子のインデックス，$k$ は時間方向の格子のインデックスである．この反復空間と依存性を図示すると図 1 のようになる．図 1 には並列計算を想定した領域分割を破線で示してある．

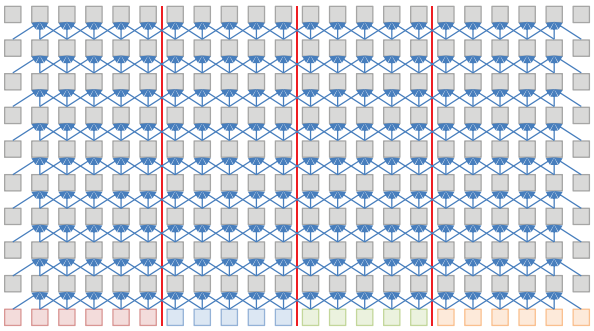
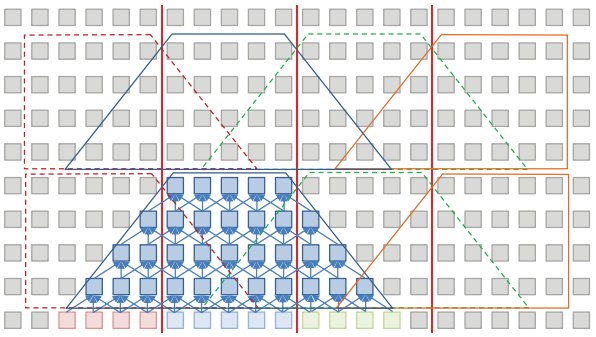


**図 1** 有限差分法による 1 次元熱伝導方程式の計算グラフ

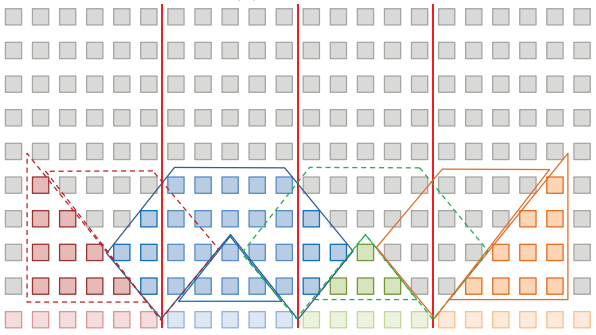
図 2 に 3 種類の時空間タイリング方式を示す．図 2 (a) は，本稿では**台形方式**と呼ぶ手法を示している．これは図 1 で与えられた領域分割に基づき，示された時空間格子を 1 回の通信で並列計算する時空間タイリングである．各領域について，図の最上部にあたる最後の時間ステップの格子点の値が依存している値の全体が台形で示されている．そこで，

(1) 最初に，通信により，各プロセッサが台形の底辺にあたるデータを受け取る．
(2) 次に，各プロセッサが台形の範囲内の格子点での値を計算する．

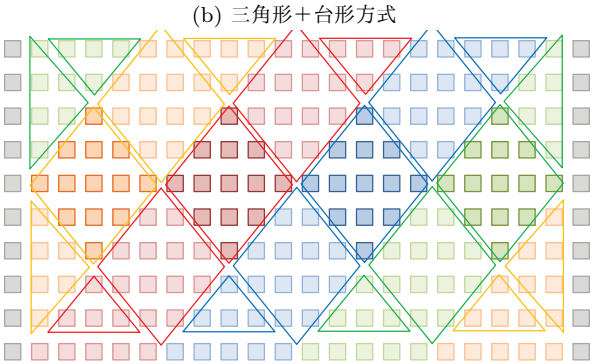
という 2 ステップの処理を行うことで，すべての計算を行うことができる．この手法の特徴は，通信は 1 回で済むが，計算の重複が多いことである．これは後述のように行列冪カーネルに容易に拡張でき，PA1 と呼ばれている．



(a) 台形方式



(b) 三角形＋台形方式



(b) 菱形方式

**図 2** 1 次元ステンシル計算の時空間タイリング

図 2 (b) は，本稿では**三角形＋台形方式**と呼ぶ手法を示している．これもやはり図 1 で与えられた領域分割に基づき，1 回の通信で計算を行う．各領域について，それぞれの領域の最初の値（三角形の底辺にあたる）から，可能な限りの計算を行う．すると，図の下の方に並んだ三角形の範

囲が計算できる．その後は図 2 (a) の台形のうち，まだ計算されていない部分を各プロセッサで計算する．すなわち
(1) 各プロセッサで三角形の範囲を計算する．
(2) 必要な通信を行う．
(3) 各プロセッサで台形の残りの部分を計算する．
という 3 ステップの処理により，格子点のすべての値を計算する．台形方式に比べて通信回数は増やすことなく，計算の重複を減らしている．これも後述のように行列冪カーネルに容易に拡張でき，PA2 と呼ばれている．

図 2 (c) は，本稿では**菱形方式**と呼ぶ手法を示している．最初は各領域について，初期値から計算できるところまで計算して，三角形の時空間領域を計算するところまでは三角形＋台形方式と同様である．この次の通信においては，各プロセッサが左隣のプロセッサに計算結果を送る．通信の後で，各プロセッサが持っている情報から計算できる限りの範囲を計算すると，三角形の上に載っている菱形の領域となる．最初の三角形は菱形の上半分にほかならない．
(1) 各プロセッサで三角形の部分を計算する．
(2) 計算結果を左隣のプロセッサに送信する．
(3) 受け取ったデータで可能になる菱形の領域を計算する．
(4) ステップ 2 と 3 を必要なだけ繰り返す.
というステップにより計算が行われる．この手法の特徴は，計算の重複は一切生じないが，通信は必ずしも 1 回では済まないこと，また各プロセッサが担当する領域が移動していくことである．これが本稿で提案する行列冪カーネルである．

上述の 3 点ステンシルの時空間タイリングは，三重対角行列に対する行列冪カーネルに直接対応する．

### 2.3 既存の行列冪カーネル

まず，本稿の主要部分の説明にも必要なため，以下の定義を行う．今まで通り $x^{(k)} = A^k x$ とする．ベクトル $x^{(k)}$ の第 $i$ 要素は $x_i^{(k)}$ と表される．以下，簡単のために行列 $A$ は全プロセッサがデータを持っていると仮定し，行列のデータ分散は考慮しない．

#### 2.3.1 スカート

まず $x = x^{(0)}$ から $x^{(1)} = Ax$ を計算することを考える．$x_i^{(1)}$ は $A$ が疎行列であることを明示すると

$$x_i^{(1)} = \sum_{j|A_{ij}\neq 0} A_{ij} x_j^{(0)}$$

のようにあらわされる．そこでここで出てきた $j$ の集合を

$$Sk^{(1)}(i) = \{j \mid A_{ij} \neq 0\}$$

と置き，要素 $i$ の**レベル 1 のスカート**と呼ぶ．すなわち

$$x_i^{(1)} = \sum_{j\in Sk^{(1)}(i)} A_{ij} x_j^{(0)}$$

であり，$x_i^{(1)}$ の計算が依存している $x^{(0)}$ の添え字の集合を示している．

また，$x^{(1)}$ の複数の要素 $i \in I$ について計算するために参照する必要がある添え字の集合を

$$Sk^{(1)}(I) = \cup_{i\in I} Sk^{(1)}(i)$$

とする．添え字の集合 $I$ は領域とみなす．そして $Sk^{(1)}(I)$ は領域 $I$ の**レベル 1 のスカート**と呼ぶ.

次に $x_i^{(2)}$ を計算することを考える．この計算が依存している値は $x_j^{(1)} \mid j \in Sk^{(1)}(i)$ であるが，これらの値がさらに依存している値は $x_j^{(0)}$ のうち $j \in Sk^{(1)}(Sk^{(1)}(i))$ となるものとなる．そこで

$$Sk^{(2)}(i) = Sk^{(1)}(Sk^{(1)}(i))$$

と定義する．すなわち，$x^{(0)}$ のうち $Sk^{(2)}(i)$ の値がすべてわかっていれば，通信することなく $x_i^{(2)}$ が求められる．$Sk^{(2)}(i)$ を，要素 $i$ の**レベル 2 のスカート**と呼ぶ.

先と同様にして，$x^{(2)}$ の複数の要素 $i \in I$ を計算するには

$$Sk^{(2)}(I) = Sk^{(1)}(Sk^{(1)}(I))$$

の値がわかっていればよい．$Sk^{(2)}(I)$ を，領域 $I$ の**レベル 2 のスカート**と呼ぶ.

同様にして，要素 $i$ の**レベル $k+1$ のスカート**である

$$Sk^{(k+1)}(i) = Sk^{(1)}(Sk^{(k)}(i))$$

および，領域 $I$ の**レベル $k+1$ のスカート**である

$$Sk^{(k+1)}(I) = Sk^{(1)}(Sk^{(k)}(I))$$

が，$k = 2, 3, \ldots$ に対して再帰的に定義される．これがステンシル計算における時空間タイリングでの台形に対応する．

本稿では，非零パターンが対称で対角要素が非零の行列を仮定している．その結果，

$$Sk^{(k+1)}(I) \supseteq Sk^{(k)}(I)$$

が成り立つ.

#### 2.3.2 錐

次に，ステンシル計算における時空間タイリングでの三角形に対応する概念を導入する．我々はこれを**錐**と呼んでいる．

いま，あるプロセッサが領域 $I$ に対応する $x^{(0)}$ の要素，つまり $x_j^{(0)} \mid j \in I$ の値を知っているとする．もしある $i$ に対して

$$Sk^{(1)}(i) \subseteq I$$

が成り立てば，$x_i^{(1)}$ を計算するために必要な情報が揃っており，他のプロセッサと通信することなく $x_i^{(1)}$ を計算することができる．そこで，このプロセッサが手持ちのデータ

から計算できる $x^{(1)}$ の要素の集合は

$$Cn^{(1)}(I) = \{i \mid Sk^{(1)}(i) \subseteq I\}$$

ということになる．これを領域 $I$ の**レベル 1 の錐**と呼ぶ．
同様に，もしある $i$ に対して

$$Sk^{(2)}(i) \subseteq I$$

が成り立てば，$x_i^{(2)}$ も通信することなく計算することができる．このように考え，

$$Cn^{(k)}(I) = \{i \mid Sk^{(k)}(i) \subseteq I\}$$

を領域 $I$ の**レベル $k$ の錐**と呼ぶ．

本稿では，非零パターンが対称で対角要素が非零の行列を仮定している．その結果，

$$Cn^{(k+1)}(I) \subseteq Cn^{(k)}(I)$$

が成り立つ．すなわち，

$$\cdots Cn^{(2)}(I) \subseteq Cn^{(1)}(I) \subseteq I \subseteq Sk^{(1)}(I) \subseteq Sk^{(2)}(I) \cdots$$

となっている．

#### 2.3.3 既存手法 PA1 および PA2

これらの準備のもと，PA1 と PA2 は以下のように説明することができる．

PA1 は台形方式に対応するので，スカートのみを用いる．プロセッサ $p$ は領域 $I_p$ を担当しているとすると，

(1) プロセッサ $p$ が $x_j^{(0)} \mid j \in Sk^{(K)}(I_p)$ を持つように，必要な通信を行う．
(2) プロセッサ $p$ はまず $x_j^{(1)} \mid j \in Sk^{(K-1)}(I_p)$，次に $x_j^{(2)} \mid j \in Sk^{(K-2)}(I_p)$，と計算してゆき，最後に $x_j^{(K-1)} \mid j \in Sk^{(1)}(I_p)$ から $x_j^{(K)} \mid j \in I_p$ を計算する．

のように処理を行う．通信は計算の前に最初に 1 回だけ行われる．スカートだけを用いることから，**スカート手法**と呼ぶこともできる．

PA2 は三角形＋台形方式に対応するので，錐とスカートの両方を用いる．先と同様の仮定のもと，

(1) プロセッサ $p$ は自前のデータ $x_i^{(0)} \mid i \in I$ から計算できる，$x_i^{(1)} \mid i \in Cn^{(1)}(I)$ を最初に，次に $x_i^{(2)} \mid i \in Cn^{(2)}(I)$ を計算し，順次 $x_i^{(K)} \mid i \in Cn^{(K)}(I)$ まで計算する．（ある $k$ では $Cn^{(k)}(I) = \emptyset$ となりうる．このときには実際には何の計算もしない．）
(2) プロセッサ間で必要な通信を行う．説明を簡単にするため，ステップ 1 で計算されたすべての要素を $Dn$ として，全プロセッサが持つことにしてもよい．ベクトル $x^{(k)}$ のうち計算された範囲の添え字の集合は

$$Dn(k) = \cup_p Cn(k)(I_p)$$

である．

(3) プロセッサ $p$ は，自分で計算する必要がある $x_i^{(K)} \mid i \in I_p$ およびそのスカートのうち，計算されていない部分，すなわち

$$x_j^{(k)} \mid j \in Sk^{(K-k)}(I_p) - Dn(k) \qquad k = 1, 2, \ldots, K$$

を計算する．

のように処理を行う．錐を 1 回，スカートを 1 回使うことから，**錐＋スカート手法**と呼ぶこともできる．

## 3. 一般化菱形行列冪カーネル

本稿では，ステンシル計算における菱形方式に対応する行列冪カーネルを考える．

### 3.1 基本方針

これは，今までに考えてきたスカートと錐だけでは定義することが難しい．そこで，菱形方式ではプロセッサが担当する範囲が移動していくことに注目する．各プロセッサが担当する領域が通信するたびに変わってゆくこと，そして計算の重複を行わないこと，この 2 点に注目して，以下のような手法を考える．

**領域分割 $I^{(j)}$**

計算の重複を行わないため，領域は重なりのない部分領域に分割してプロセッサに割り当てる必要がある．通信を 1 回行うごとに担当領域が変わることを想定して，$j$ 回目の通信後にプロセッサ $p$ が担当する領域を $I_p^{(j)}$ とする．初期状態での領域分割では，プロセッサ $p$ は $I_p^{(0)}$ を担当している．

**計算済み添え字集合 $Dn^{(j)}$**

また，計算の重複を行わないため，通信の前に計算が終わった範囲を把握しておく必要がある．ベクトル $x^{(k)}$ のうち，$j$ 回目の通信の前までに，どこかのプロセッサで計算された添え字の範囲を $Dn^{(j-1)}(k)$ とする．前節の PA2 の説明で出てきた $Dn(k)$ は $Dn^{(0)}(k)$ に対応する．

**アルゴリズムの方針**

これらを基にして，次のような方針でアルゴリズムを考える．

- 必要な通信は行われるとして，$j$ 回目の通信の後は $Dn^{(j-1)}(k)$ の要素の値はどのプロセッサでも参照できるものとする．
- プロセッサ $p$ は自分の担当領域 $I_p^{(j)}$ の要素しか計算しない．そして，「計算できる限りの要素」は計算する．

この「計算できる限りの要素」が何であるか，次に考える．

### 3.2 一般化した錐 $Cn$

第 $j$ 回目の通信の後，ベクトル $x^{(k)}$ のうち，プロセッサ $p$ の担当領域で計算できる要素 $i$ は次の通りである．なお，記号の定義から，これらの $i$ の範囲は

$$i \in (Dn^{(j)}(k) - Dn^{(j-1)}(k)) \cap I_p^{(j)}$$

に他ならない．すでに計算されている範囲は重複して計算することはしないから，添え字 $i$ は $Dn^{(j-1)}(k)$ に含まれないからである．

ベクトル $x^{(k)}$ はもちろん $k$ が小さい方から順に計算する．まず，$k$ として $I_p^{(j)} - Dn^{(j-1)}(k) \neq \emptyset$ となるような最小の $k$ を考える．すなわち，担当領域 $I_p^{(j)}$ のうち，まだ計算されていない要素が残っているような $x^{(k)}$ のうち $k$ が最小のものを考える．このとき，$x^{(k)}$ の要素 $i$ がプロセッサ $p$ で計算できるためには，

$$i \in I_p^{(j)} \text{ かつ } Sk^{(1)}(i) \subseteq Dn^{(j-1)}(k-1)$$

である必要がある．これを満たす添え字 $i$ の集合を $Cn_p^{(j)}(k)$ とする．すなわち

$$Cn_p^{(j)}(k) = \{i \in I_p^{(j)} \mid Sk^{(1)}(i) \subseteq Dn^{(j-1)}(k-1)\}$$

である．

次のベクトル $x^{(k+1)}$ について考える．プロセッサ $p$ は，ベクトル $x^{(k)}$ のうち，通信前に得られていた値 $Dn^{(j-1)}(k)$ および前段で新たに計算した値 $Cn_p^{(j)}(k)$ を知っている．したがって，$x^{(k+1)}$ のうち計算できる添え字の範囲は

$$\begin{aligned} &Cn_p^{(j)}(k+1) \\ &= \{i \in I_p^{(j)} \mid Sk^{(1)}(i) \subseteq Dn^{(j-1)}(k) \cup Cn_p^{(j)}(k)\} \quad (1) \end{aligned}$$

となる．これを $k+2, k+3, \ldots, K$ まで繰り返すことで，$Cn_p^{(j)}(k+2)$ から $Cn_p^{(j)}(K)$ までが求まる．$Cn_p^{(j)}(0)$ から $Cn_p^{(j)}(k-1)$ までは空集合と定義することで，式 (1) はすべての $k$ について成り立つ．

すなわち，これらの $Cn_p^{(j)}(k)$ は，プロセッサ $p$ において通信前に計算されていた値 $Dn^{(j-1)}$ から計算できる範囲を示している．これは前述の錐の一般化になっている．すなわち，前節での錐は，参照できる値が $x^{(0)}$ だけであったところであるが，ここでは一般に $x^{(k)}$ のうち添え字の範囲が $Dn^{(j-1)}(k)$ の要素は参照できるとして，通信することなしに計算できる範囲である．これがステンシル計算における菱形タイルに相当するものである．これら $Cn_p^{(j)}$ を $Dn^{(j-1)}$ **上の領域** $I_p^{(j)}$ **における錐**または**一般化した錐**と呼ぶことにする．

以上の説明はいちおう正しいが，ややわかりにくい．一般化した錐は以下のようにより簡明な方法で計算することができる．

### 3.3 錐の高さ

行列冪カーネルの目標は通信削減である．すなわち，少ない通信回数でこの計算を実現したい．本稿ではある正整数 $K$ が与えられていて，$x^{(K)}$ まで計算することを考えている．第 $j$ 回目の通信を行ったあとにする計算が，一般化された錐 $Cn_p^{(j)}$ であるから，この計算量ができるだけ大きい方がよい．「錐」という幾何学的なイメージからすると，錐が高いほど良い，と言い直すことができる．

第 $j$ 回目の通信の後に錐 $Cn_p^{(j)}$ がどこまで達するかは，その通信直前の計算済み領域である $Dn^{(j-1)}$ によって決まる．$Dn^{(j-1)}$ のすべての要素が複雑にからまりあって決まるわけではなく，実は，領域 $I_p$ のいわゆる袖領域だけで決まる．

**添え字の高さ** $h_i^{(j)}$

第 $j$ 回目の通信の前 $Dn^{(j-1)}$ を考える．添え字 $i$ を固定すると，$x_i^{(0)}, x_i^{(1)}, \ldots, x_i^{(K)}$ のうち，あるところまでは計算済みであり，あるところからは未計算である．すなわち，ある $h$ があって，$x_i^{(0)}$ から $x_i^{(h)}$ までは計算済み，すなわち

$$i \in Dn^{(j-1)}(0), \ \ldots, \ i \in Dn^{(j-1)}(h)$$

であり，$x_i^{(h+1)}$ から $x_i^{(K)}$ までは未計算，すなわち

$$i \notin Dn^{(j-1)}(h+1), \ \ldots, \ i \notin Dn^{(j-1)}(K)$$

である．このようにきっぱりわかれるのは，疎行列 $A$ の対角要素がすべて非零であるという仮定によっている．このような $h$ を添え字 $i$ の**高さ**と呼び，通信の回数 $j$ と添え字 $i$ を明示してその高さを $h_i^{(j-1)}$ と書くことにする．

第 $j$ 回目の通信の後の計算後の高さは $h_i^{(j)}$ であるが，これは次のように求めることができる．その前にいくつか確認をしておく．

**隣接する添え字とその高さ**

添え字 $i$ に対して添え字 $\hat{i}$ が $\hat{i} \in Sk^{(1)}(i)$ を満たすとき，$i$ と $\hat{i}$ は**隣接している**という．行列の非零パターンが対称であるという仮定から，$i$ と $\hat{i}$ が隣接しているとき，$\hat{i}$ と $i$ も隣接している．

添え字 $i$ と $\hat{i}$ が隣接しているとき，それらの高さ $h_i^{(j-1)}$ と $h_{\hat{i}}^{(j-1)}$ の差は高々 1 である．すなわち

$$|h_i^{(j-1)} - h_{\hat{i}}^{(j-1)}| \leq 1$$

である．これは，$x_i^{(k)}$ を計算するためには $x_{\hat{i}}^{(k-1)}$ が必要であり，また $x_{\hat{i}}^{(k)}$ を計算するためには $x_i^{(k-1)}$ が必要であるためである．

**錐の高さの別表現**

プロセッサ $p$ の担当領域 $I_p^{(j)}$ における錐 $Cn_p^{(j)}$ を求めるためには，領域 $I_p^{(j)}$ のいわゆる袖領域にある要素が参照される．この袖領域は

$$H_p^{(j)} = Sk^{(1)}(I_p^{(j)}) - I_p^{(j)}$$

と表すことができる．

プロセッサ $p$ から見ると，袖領域 $H_p^{(j)}$ に含まれる要素 $\hat{i}$ については，高さ $h_{\hat{i}}^{(j-1)}$ までの情報を直前の通信で得ているが，自分では計算しないので，それより新しい情報は次の通信を行うまで得られない．従って，$\hat{i}$ に隣接する要素 $i \in I_p^{(j)}$ は高さが高々 $h_{\hat{i}}^{(j-1)} + 1$ までしか計算できな

い．すなわち

$$h_i^{(j)} \leq h_{\hat{i}}^{(j-1)} + 1$$

である．同様にして，$i$ に隣接する要素 $i' \in I_p^{(j)}$ があったとき，$i'$ の高さは $i$ の高さより 1 高いところまでしか計算できない．すなわち

$$h_{i'}^{(j)} \leq h_i^{(j)} + 1 \leq h_{\hat{i}}^{(j-1)} + 2$$

ということになる．要素 $i'$ は袖領域からの距離（グラフとしての頂点間の距離）が 2 以下の要素であって，それが上式の最後の 2 の由来となっている．同様にして考えると，袖領域の要素 $\hat{i}$ からの距離が $d(i,\hat{i})$ の要素 $i$ について，

$$h_i^{(j)} \leq h_{\hat{i}}^{(j-1)} + d(i,\hat{i})$$

ということが言える．そして，この制約条件を満たす範囲では実際に計算ができる．すなわち

$$h_i^{(j)} = \min\{h_{\hat{i}}^{(j-1)} + d(i,\hat{i}) \mid \hat{i} \in H_p^{(j)}\} \qquad (2)$$

が成り立つ．

この高さが錐の高さを決める．この式からわかるのは，袖領域の要素の高さが高いほど，錐の高さが高くなるということである．

**計算範囲の底**

計算された範囲 $Dn^{(j)}$ において，もっとも高さの低い要素を**底**と呼ぶことにする．上記の議論を参照すると，1 回の通信のあとの計算で，底の高さは少なくとも 1 は上がることが確認できる．すなわち，通信と疎行列ベクトル積とを交互に行う通常のアルゴリズムと少なくとも同じ回数の通信で計算できる．すなわち，冪指数 $K$ までの行列冪カーネルを計算するために必要となる通信回数は $K$ 以下である．

### 3.4 領域分割

以上のようにアルゴリズムを定式化することにより，領域分割 $I_p^{(j)}$ さえ決めれば錐の高さが決まることになる．そして，袖領域の要素の高さが高いほど錐は高くなる．すなわち，領域分割の境界ができるだけ高いところを通るように，領域分割を定めたい．直感的な言い方をすると，「尾根伝い」に領域の境界を定めるのがよい．

逆にいうと，高さが低い頂点を結ぶ辺を切るような領域分割を避けたいと考えることもできる．この考え方を使うと，既存のグラフ分割ソフトウェアを活用することができる．我々は METIS を利用することにした．

METIS では，グラフの頂点と辺に重みをつけることができる．METIS は，部分グラフの頂点の重みの総和が同程度で，部分グラフをまたぐ辺の重みの総和ができるだけ小さくなるようなグラフ分割を求めてくれる．

そこで，我々は高さが低い頂点を結ぶ辺の重みを重くして，それが領域分割の境界になりにくいように設定した．そのような重みづけには様々なものが考えられるが，いくつかの予備的実験ののち以下のように設定した．すなわち頂点 $i$ と $j$ を結ぶ辺の重みを

$$w_{ij} = \frac{10^6}{h_i + h_j - 2h_{\min} + 1}$$

とした．ただし $h_{\min}$ はすべての頂点の高さの最小値である．もし辺の両側の頂点の高さがいずれも $h_{\min}$ であるとき，重みは $10^6$ となる．頂点の高さが高くなると，幾何的に重みが減少する．なお，使用したプログラムは gpmetis であり，頂点の重みはすべて 1 とした．また，オプションとして ufactor=1000 とした（これは領域間で 2 倍までの頂点数の違いを許容する）．

### 3.5 反復可能な行列冪カーネル

我々は Krylov 部分空間法のような反復法に用いることを想定して行列冪カーネルを設計している．この場合，行列冪カーネルを何度も繰り返して用いることになる．（むしろ，繰り返し用いなければ領域分割のコストがかかりすぎて役に立たない．）

このため，計算する冪指数 $K$ を定めて領域分割を固定し，行列冪カーネル $x^{(k)} = A^k x$ $(k = 1, 2, \ldots, K)$ を何度も繰り返し再利用する方針を取る．一度行列冪カーネルを実行した後，次に行列冪カーネルを実行する際には，最初の領域分割から再スタートすることになる．そこで，行列冪カーネルを終わったところで，最初の領域分割に戻った状態になるのが望ましいであろうと我々は考えた．

これを実現するには，行列冪カーネルが最初の領域分割とは異なる領域分割で終わった後，別途データ再分散を行って元の領域分割に戻すという方式も考えられる．しかし，我々は行列冪カーネルの中に元の領域分割に戻す仕組みを入れておくほうがよいと想定した．そこで我々は，これまでに説明してきた錐だけを用いたアルゴリズムではなく，スカートも利用した以下のアルゴリズムを提案する．

まず，冪指数 $K$ は与えられたものとして，通信を行う回数 $C$ を選んでおく．$C$ は $K$ 未満の正整数である．

(1) 通信回数 $j = 0$ とする．
(2) 領域分割 $(I_p^{(j)})$ を定める．
(3) 錐 $Cn^{(j)}$ を計算する．
(4) 通信回数 $j$ に 1 加え，$j < C$ ならステップ (2) に戻る．
(5) 最後の領域分割 $I^{(C)}$ は初期領域分割 $I^{(0)}$ と同じとする．
(6) 各プロセッサ $p$ に対して領域 $I_p^{(C)}$ のスカート $Sk^{(k)}$ を求め，計算がされていない部分（$Sk^{(K-k)}(I_p^{(C)}) - Dn^{(C-1)}$）を計算する．

すなわち，最後の通信の後は初期領域分割に戻る．そして各プロセッサが担当領域の（錐ではなく）スカートのうち，未計算部分を計算する．こうすることにより，必ず $C$ 回

以下の通信で行列冪カーネルが実現できる．また，行列冪カーネルが終了した時点で最初の領域分割に戻っており，反復して行列冪カーネルを実行することができる．

ただし，最後はスカートで計算範囲を決めているため，ここで計算の重複が発生しうる．一般的に通信回数 $C$ を大きくすると，計算の重複を生じない錐による計算が増加し，最後のスカートにおける計算の重複が減少すると考えられる．通信の回数増加による所要時間増大と，計算の重複による所要時間増大は，計算機ごとに具体的なコストが異なると考えられる．そこで，適切な $C$ を選んでやることにより，通信と計算のコストのトレードオフを実現することができると考えられる．逆にいうと，最適な $C$ がいくつかということは一概に決めることはできない．

## 4. 予備実験

本節ではいくつかの行列に関して提案手法を適用し，その効果を検証する．研究目的は超大規模並列計算であるが，ここでは提案手法の動作を確認するため小規模な問題に適用している．

### 4.1 実験内容

用いた行列は表 1 の通りである．なお，Shuttle_eddy と Cage10 はいわゆる Florida 大学疎行列コレクション [10] から使用した．

**表 1** 実験に用いた行列

| 行列 | サイズ | 非零要素数 |
|---|---|---|
| 2 次元メッシュ ($100 \times 100$) | 10,000 | 19,800 |
| 3 次元メッシュ ($25 \times 25 \times 25$) | 15,625 | 45,000 |
| Shuttle_eddy | 10,429 | 103,559 |
| Cage10 | 11,397 | 150,645 |

プロセッサにあたる領域分割数は 25 とした．従って，1 プロセッサあたり 400 から 600 程度というごく少数のノードを担当するような条件となっている．

以下の実験では，実際に行列冪カーネルは実行していない．行っていることは，指定された回数の領域分割を行い，錐および最後のスカートを求めている．最後のスカートにおける重複計算を調べることで，計算の重複量を求めることができる．

また，以下のようにして通信量も仮の値を求めている．実際に行列冪カーネルを並列実装する際には，各領域をプロセッサに割り当て，通信前にすでにプロセッサが持っているデータについては通信をする必要がないとするべきである．従って，どの領域をどのプロセッサに割り当てるかによって，通信量が変わってくる．最後には初期領域分割に戻るとしていることにも注意すると，領域のプロセッサ割り当ては離散的な最適化問題になっていることがわかる．厳密な最適解を求めることは現実的な計算量にならないと予想しているが，この部分は今後の課題として，今回は通信を省略できる部分がまったくない最悪の割り当てにおける通信量を求めることにより，通信量の参考になる値を出している．

### 4.2 実験結果

結果として，従来手法である PA1, PA2 とともに，提案手法をいくつかの通信回数 $C$ に対して示し，また通信削減しない古典手法についても示す．指標としては，通信回数，計算量（計算重複の量を示す），通信量，錐の底の高さ，平均錐高（最後の通信直前の高さの平均）を示す．なお，通信量については前述の通り過大評価になっており，最大 2 倍以上の過大評価にもなりうるため，参考値であることを明示するためイタリックで示す（PA1, PA2 および古典は正確に計算されている）．なお，従来手法 PA2 は $C = 1$ の提案手法に他ならない．また，比較のため，PA2 の冪指数を低くしたものも示した．たとえば PA2 で $K = 5$ のカーネルを 2 回呼ぶと $K = 10$ の冪カーネルが 2 回の通信で実行できる．このほか，$K = 4$ のカーネルを 2 回と $K = 2$ のカーネルを 1 回呼ぶ組み合わせ，$K = 3$ のカーネルを 3 回と $K = 1$ のカーネルを 1 回呼ぶ組み合わせについて調べ，これらを PA(2), PA(3), PA(4) として示した．カッコ内が通信の回数を示している．

表 2 に 2 次元メッシュによる結果を示す．3 回目の通信から底が上がってゆき，平均錐高も順調に伸びている．$C = 3$ ですでに 98%の計算が終わっており，計算の重複はほとんどない．古典手法に比べて大幅に有利である．また，PA1, PA2 に比べると，計算の重複が効果的に削減されていることも確認できる．また，PA2 を反復する方法に比べて計算量の増大が抑えられていることがわかる．

**表 2** 2 次元メッシュ ($100 \times 100$) による結果 ($K = 10$)

| 手法 | 通信回数 | 計算量 | 通信量 | 錐の底 | 平均錐高 |
|---|---|---|---|---|---|
| PA1 | 1 | 18.4 | 1.96 | - | - |
| PA2 | 1 | 15.2 | 1.57 | 0 | 3.34 |
| $C = 2$ | 2 | 11.4 | *3.58* | 1 | 7.60 |
| $C = 3$ | 3 | 10.01 | *4.28* | 5 | 9.83 |
| $C = 4$ | 4 | 10.00 | *6.00* | 8 | 10.00 |
| 古典 | 10 | 10.00 | 1.72 | - | - |
| PA(2) | 2 | 13.5 | 1.87 | - | - |
| PA(3) | 3 | 12.3 | 1.78 | - | - |
| PA(4) | 4 | 11.6 | 1.59 | - | - |

表 3 に 3 次元メッシュによる結果を示す．底が通信回数よりも高くなるのは遅れているが，平均錐高は通信回数の 2 倍程度あり，$C = 5$ と古典を比較すると，ほぼ通信回数を半減できていることがわかる．2 次元メッシュよりも効果が低いのは，各領域の 1 次元あたりの長さが短くなっているからである．PA1, PA2 に比較すると，計算の重複が効果的に削減できている．実は PA2 を反復した方が計算

の重複が少ないという結果になっている．これは領域分割のチューニングが不足している可能性もあるが，一般的には計算の重複を完全に許さない方針よりも，多少許したほうが結果的によいという可能性がある．

**表 3** 3 次元メッシュ ($25 \times 25 \times 25$) による結果 ($K = 10$)

| 手法 | 通信回数 | 計算量 | 通信量 | 錐の底 | 平均錐高 |
|---|---|---|---|---|---|
| PA1 | 1 | 41.1 | 8.00 | - | - |
| PA2 | 1 | 35.7 | 8.64 | 0 | 1.03 |
| $C = 2$ | 2 | 25.4 | *9.87* | 1 | 3.13 |
| $C = 3$ | 3 | 17.8 | *10.04* | 2 | 5.24 |
| $C = 4$ | 4 | 13.2 | *10.33* | 3 | 7.08 |
| $C = 5$ | 5 | 11.0 | *10.81* | 4 | 8.49 |
| $C = 6$ | 6 | 10.23 | *11.46* | 5 | 9.44 |
| $C = 7$ | 7 | 10.02 | *12.55* | 7 | 9.87 |
| 古典 | 10 | 10.00 | 5.06 | - | - |
| PA(2) | 2 | 21.5 | 6.52 | - | - |
| PA(3) | 3 | 17.1 | 5.99 | - | - |
| PA(4) | 4 | 14.8 | 5.19 | - | - |

表 4 に行列 shuttle_eddy による結果を示す．底は 3 回目の通信から順調に上がるようになっている．平均錐高の伸びは目覚ましく，$C = 4$ 程度でほぼ計算が終了している．この例では PA1, PA2 に比べて計算の重複が効果的に削減されているとともに，PA2 を反復する方式よりも計算の重複が少ない．

**表 4** shuttle_eddy による結果 ($K = 10$)

| 手法 | 通信回数 | 計算量 | 通信量 | 錐の底 | 平均錐高 |
|---|---|---|---|---|---|
| PA1 | 1 | 24.7 | 3.66 | - | - |
| PA2 | 1 | 19.7 | 3.37 | 0 | 2.25 |
| $C = 2$ | 2 | 14.4 | *5.40* | 1 | 5.12 |
| $C = 3$ | 3 | 11.0 | *6.03* | 2 | 8.08 |
| $C = 4$ | 4 | 10.15 | *6.83* | 4 | 9.51 |
| $C = 5$ | 5 | 10.01 | *8.21* | 6 | 9.93 |
| 古典 | 10 | 10.00 | 2.37 | - | - |
| PA(2) | 2 | 15.6 | 3.11 | - | - |
| PA(3) | 3 | 13.4 | 2.89 | - | - |
| PA(4) | 4 | 12.3 | 2.52 | - | - |

表 5 に行列 cage10 による結果を示す．この行列は特別であり，底は 1 回の通信で最低限である 1 しか高くならない．平均錐高もほとんど通信回数と並行しており，伸び悩んでいる．PA1, PA2 は大量の重複計算を生んでいるが，少ない冪指数で PA2 を繰り返す方式は比較的よい結果であるように見える．この例では $K = 2$ を 5 回繰り返す PA(5) も記載してみたが，意外なことに PA(4) よりも計算重複が増えている．PA(4) は $K = 1$ つまり古典での計算を 1 回行っており，それが計算重複の削減に効いている可能性がある．

これらの例からわかるように，行列によって提案手法の有効性が異なる．メッシュあるいは有限要素法のような離散化から生じた行列は，各頂点に隣接する頂点が少なく，また直径が比較的大きい．このような行列では疎行列ベクトル積において隣接通信に近い少ない通信で足りることが多く，効率的な並列化が可能である．そのような行列に対しては，提案手法は有効に働く．

**表 5** cage10 による結果 ($K = 10$)

| 手法 | 通信回数 | 計算量 | 通信量 | 錐の底 | 平均錐高 |
|---|---|---|---|---|---|
| PA1 | 1 | 153.3 | 24.0 | - | - |
| PA2 | 1 | 147.7 | 24.3 | 0 | 0.24 |
| $C = 2$ | 2 | 121.3 | *27.4* | 1 | 1.37 |
| $C = 3$ | 3 | 97.1 | *29.7* | 2 | 2.44 |
| $C = 4$ | 4 | 74.0 | *32.3* | 3 | 3.49 |
| $C = 5$ | 5 | 54.3 | *34.4* | 4 | 4.49 |
| $C = 6$ | 6 | 37.0 | *35.3* | 5 | 5.49 |
| $C = 7$ | 7 | 23.9 | *33.6* | 6 | 6.49 |
| 古典 | 10 | 10.0 | 15.4 | - | - |
| PA(2) | 2 | 73.5 | 38.4 | - | - |
| PA(3) | 3 | 45.0 | 35.3 | - | - |
| PA(4) | 4 | 30.4 | 29.8 | - | - |
| PA(5) | 5 | 35.4 | 26.3 | - | - |

しかし cage10 のような行列では，提案手法の効果がほとんどない．おそらく直径がかなり小さく，低次数の行列冪がほとんど密行列になるような例であると考えられる．そのような行列では，行列冪カーネルによって通信回数を削減することは，そもそも困難であると考えられる．

なお，通信量については今回は参考値にしかならないが，提案手法は古典手法や PA1, PA2 に比べて通信量が多くなることを著者は予想しており，それを示唆するようなデータとなっている．

## 5. まとめと今後の課題

本稿では，行列冪カーネルについて著者が [11] で示した手法を詳述した．この手法は既存手法 PA1, PA2 とは異なり計算の重複が削減できる．本稿ではアルゴリズムの詳細，特に，領域分割の具体的手法を説明した．小規模な行列に適用した結果，いくつかの行列では期待に沿うように，少ない計算削減で，通信回数を削減することができた．しかし行列によってはまったく通信削減をできず，古典的な手法で計算するのがよいと考えられる．

今後の課題は多岐にわたる．まず本稿で説明した範囲の課題を挙げる．本稿では領域分割に用いる辺の重みとして一つの式を出したが，無数に考えられる式のうちこれが特別に優れているということは本稿では主張していない．むしろよりよい重み式があると期待され，その開発が望まれる．また，そもそも従来手法の領域分割と，我々の行列冪カーネルにおける領域分割では目指すものが違う．行列冪カーネルに適した領域分割はどういうものか，一から検討するべきである．また，今回は領域への分割は行っているが，プロセッサへの割り当ては行っていない．これは適当

な手法を検討する必要がある．

大規模実装に向けては，さらに大きな課題が残っている．神威では 1000 万コアあるということであるが，その全系で処理される疎行列は少なくとも 100 億ノード，あるいは 1000 億ノードという規模になる．このような超大規模グラフに対して良質な領域分割を実現できるかどうかがそもそも問題である．また，今回は計算結果および行列そのもののデータ構造やデータ分散，通信バッファリングなどについて言及していない．これら多数の課題を解決しなければ，実際の不規則疎行列の冪カーネルは実現しない．

また，スーパーコンピュータではノード間の並列性とノード内の並列性，さらに命令レベル並列性や SIMD 並列性がある．ノードに割り当てられた領域をさらに分割してノード内のコアに分散するか，逆にノード内のコアに割り当てられた領域を融合してノードの担当とするなどの手法も必要である．また，通信遅延隠蔽も基本的な技術であるが，今回の手法ではそれが適用できない．「計算できるだけ計算する」という方針がその理由である．計算できるけれども一部は計算しないで残しておく，という仕組みを導入することにより，通信遅延隠蔽の適用が可能になるが，その詳細は今後検討する必要がある．

最後に，今回は非零パターンが対称で対角要素が非零であるという仮定をおいた．これが満たされない場合には，まったく異なる手法が必要となる可能性がある．また，今回の実験でも示されたように，計算重複は完全に排除するのが最適とは限らず，ある程度計算重複を許した方が通信回数・通信量と計算量の適切なバランスが得られる可能性がある．しかしこれについても現時点では検討を開始すらしていない．

このように，今後の課題は非常に多い．むろん著者自身もこれらの課題に取り組んでゆくが，本稿を読んで多くの方に興味を持っていただけると幸いである．

**謝辞**

本研究の一部は JST CREST「進化的アプローチによる超並列複合システム向け開発環境の創出」および科研費 15H02708 と 15K12033 の補助により実施されました．

## 参考文献

[1] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-optimal Parallel and Sequential QR and LU Factorizations, SIAM J. Sci. Comput. 34(1), pp. A206–A239, 2012.
[2] T. Muranushi and J. Makino, Optimal Temporal Blocking for Stencil Computation, Procedia Computer Science, Vol. 51, 2015, pp. 1303-1312 (Proc. ICCS 2015).
[3] D. G. Wonnacott, M. M. Strout, On the Scalability of Loop Tiling Techniques, Proceedings of IMPACT 2013, Berlin, 2013, pp. 3-11.
[4] M. Hoemmen, Communication-Avoiding Krylov Subspace Methods, Ph.D thesis, UC Berkeley, 2010.
[5] 須田礼仁，本谷徹，「チェビシェフ基底共役勾配法」，2013 年ハイパフォーマンスコンピューティングと計算科学シンポジウム (HPCS 2013)，ポスター発表
[6] 須田礼仁、李聡、島根浩平、「数値的に安定性な通信削減クリロフ部分空間法」，第 19 回計算工学講演会、2014 年．
[7] Y. Kumagai, A. Fujii, T. Tanaka, Y. Hirota, T. Fukaya, T. Imamura and R. Suda, Performance Analysis of Chebyshev Basis Conjugate Gradient Method on the K Computer, Proc. 11th International Conference on Parallel Processing and Applied Mathematics (PPAM2015).
[8] I. Yamazaki, S. Rajamanickam, E. Boman, M. Hoemmen, M. Heroux, and S. Tomov: Domain decomposition preconditioners for communication-avoiding Krylov methods on a hybrid CPU/GPU cluster. Proc. SC 2014: 933-944
[9] 渡邉大地，須田礼仁，「通信削減共役勾配法における基底ベクトル拡大数の選択」第 20 回計算工学講演会，2015 年．
[10] T. A. Davis and Y. Hu, The University of Florida Sparse Matrix Collection.
[11] 須田礼仁，「一般の行列冪カーネルに向けて」，日本応用数理学会 2015 年年会．