ヘテロジニアスなハードウェア環境におけるマルチフロンタル法

47-216674 河野 奏人
指導教員 奥田 洋司 教授

To numerically solve linear equations with large sparse matrices derived by the finite element method, a solution method that can be computed in parallel is required to take advantage of their sparsity. Iterative methods are one group of available methods, but they may not converge. In contrast, direct methods can solve such equations. We can use the multifrontal method as a part of a direct method to utilize two factors mentioned above. This paper explores how to implement the method that incorporates parallelization using MPI to achieve efficient solution in a variety of environments.

Key words: Multifrontal Method, Direct Solver, Linear Equation, Finite Element Method, Message Passing Interface, Multiple Instruction Stream-Multiple Data Stream

1 緒言

有限要素法をはじめとするモデル化によって得られる，疎行列を係数とする線形方程式を対象に解法の実装を進めた．具体的には，行列の疎性を生かした直接法であるマルチフロンタル法の実装を行った．MPI による動的プロセス生成を利用した並列計算を可能にすることで，演算の効率化とともに異なるハードウェア環境における実行を検討した．

2 序論

2.1 モデル化

自然現象・物理現象を人間が解明するに当たっては，対象となる現象や物体の変化を，微分方程式を利用して記述しモデルを構築する．[1]計算機で解く場合，連続時間・空間を扱うことは出来ないため，離散化を行う必要がある．このとき，モデル化した複数の方程式は互いに依存するため，多元連立方程式を考えることになる．以上より，ある現象を記述したモデルは，離散化した連立方程式として，行列を用いた線形方程式の形にまとめられる．

2.2 計算機の背景

高性能計算（HPC）に用いる計算機では，階層的な並列化が必要とされる．記憶装置を中心としたアーキテクチャにより計算機を分類すると，共有メモリ型と分散メモリ型に分けられるが，後者における例として MPI の利用が挙げられる．

計算の高速化という観点では，科学計算において，GPU の利用が進展きたことが特筆される．GPU は元々，ゲーム等における描画用の装置であったが，現在では用途が多様化し，本学スーパーコンピュータでも NVIDIA 社の GPU を搭載している．本研究で特に題材とする有限要素法による構造解析へ利用した先行研究も存在する．

2.3 ソルバの背景

ハードウェアの進歩により計算機が搭載するメモリが増加した．後述するように，マルチフロンタル法を含む線形方程式の直接解法では，反復法よりもメモリ消費が増大するが，これに対応可能になったと言える．

ソルバそのものの進展としては，直接法の並列化に関する研究開発が反復法ほど進んでいない点を指摘出来る．これは，直接法は依存関係を考慮しなくてはならないという特性による．実際，疎行列線形方程式の直接法については著名ないくつかの既存ライブラリに大きく依存する．

2.4 本研究の目的

計算科学で必要となる線形方程式の解法には，直説法と反復法の 2 種類が存在する．反復法は並列化が行いやすく，方程式に表れる係数行列が疎である場合に，メモリの消費量を少なく抑えることが可能である．しかしながら，いつまでも解が得られないような行列も存在する．直接法は，並列化の難しさやメモリ消費の問題があるものの，より広い範囲で計算科学の問題に対応出来る．中でもマルチフロンタル法は，弱いとされる並列化やメモリの効率利用にも優れている．本研究では，マルチフロンタル法の独自実装を行い，既存ライブラリに依存する現状の改善を企図するとともに，特に並列化について実装と検証を行う．

3 有限要素法

3.1 有限要素法とは

構造物等の変形や熱伝導を解析する場合には，物体を連続体として扱う必要がある．しかしながら，計算機は離散的な値を扱うものであり，連続値をそのまま入出力出来ない．そこで用いられる方法の 1 つが有限要素法（FEM）である．物体中に設けた節点を用いて区切ることで有限個の要素に分割し，要素ごとに求める方程式の近似を行う．これを線形方程式に落とし込むことで，解となる節点での値を求める．構造解析においては，求める変数は変位であり，さらに関係式を用いてひずみ・応力を計算する．

3.2 1 次元の有限要素法

有限要素法においては，要素ごとに区分的に定義された関数を用いることで近似を行う．ここで，そのような関数は「ある節点上で 1」となる関数を利用する．[2]与えられた微分方程式を解くには，重み付き残差法を用いることで線形方程式に帰着させる．このとき，全体の係数行列$\boldsymbol{K}$は，各要素での係数行列$\boldsymbol{K}^e$の和として得られる．[2]有限個の要素に区切ることは，連続的な分布を表現出来るという物理的な意味だけでなく，その上で区分的に定義される関数を用いることで，全体の線形方程式の係数$\boldsymbol{K}$が容易に求まるという利点が存在する．

2 次元・3 次元要素においては，1 次元における近似を拡張し，要素のある 1 節点で 1，他の節点で 0 となる 2 変数・3 変数の関数を設定すれば良い．[2]

3.3 3 次元における 4 面体 2 次要素

構造解析で用いられる代表的な 3 次元の要素に，4 面体

2 次要素がある．1 要素は 3 角形による 4 面から構成され，また各辺について端点・中点を考慮する必要があるため，1 つの要素につき必要な節点数は 10 となる．

Abaqus では「C3D10」，FrontISTR では「342」というタイプ名をそれぞれ有しており，入力データであるメッシュファイルにおける要素定義では，各要素について「[要素番号], [節点番号 1], [節点番号 2], … , [節点番号 10]」という表記法が用いられる．

4 線形方程式の数値解法

4.1 数値解法の概要

自然界の現象をシミュレーションするに当たっては，現象を支配する法則を何らかの形で連立 1 次方程式（線形方程式）に落とし込む．線形方程式は高次元であれば人の手で解くことは困難となり，計算機を用いて解となるベクトルを求める方法が必要とされる．以下では

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1}$$

の形の線形方程式を考える．ここで$\boldsymbol{A}$は行列・$\boldsymbol{b}$はベクトルであり共に既知，$\boldsymbol{x}$は求める未知のベクトルである．数学的には$\boldsymbol{A^{-1}}$ を計算すれば求解出来るが，多次元の逆行列を求めるのは現実的でないため，以下の 2 つの手法が採用される．

4.2 直接法

直接法とは，行列演算を基にした操作を利用し，近似解を利用せず真の解を直接求める方法である．ここでは$\boldsymbol{A}$を三角行列の積の形に分解する，LU 分解という手法を紹介する．LU 分解を定式化すると式(2)となる．

$$\boldsymbol{A} = \boldsymbol{LU} \tag{2}$$

LU 分解は計算手法の違いにより 3 つに分類される．そのうち「外積形式」は，$\boldsymbol{L}$ の対角成分が全て 1 であるとして分解を行う手法で，第$\boldsymbol{k}$行の U の成分と第$\boldsymbol{k}$列の L の成分を交互に定める．このとき，分解後の$\boldsymbol{L,U}$ の値を$\boldsymbol{A}$ のデータに上書きして良いので，図 1 のアルゴリズムを採用出来る．

```
1: for k = 0 to n − 1 do
2:     akkinv = 1.0 / a_{k,k}
3:     for i = k + 1 to n − 1 do
4:         a_{i,k} = a_{i,k} × akkinv
5:         aik = a_{i,k}
6:         for j = k + 1 to n − 1 do
7:             a_{i,j} = a_{i,j} − aik × a_{k,j}
8:         end for
9:     end for
10: end for
```

Figure 1 Algorithm of outer-product form LU factorization

$\boldsymbol{A}$が対称行列である場合には，$\boldsymbol{U}$でなく下三角行列の転置$\boldsymbol{L}^{\mathrm{T}}$を用いて

$$\boldsymbol{A} = \boldsymbol{LL}^{\mathrm{T}} \tag{3}$$

とする分解法もある．これをコレスキー分解と呼ぶ．ただし，コレスキー分解では A の対角成分の平方根を計算する必要があることから，0 に近い場合のエラーを防ぐため

$$\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{T}} \tag{4}$$

という修正コレスキー分解も利用される．

式(1)・(2)を合わせると

$$\boldsymbol{Ax} = \boldsymbol{LUx} = \boldsymbol{Ly} = \boldsymbol{b} \tag{5}$$

と式変形が出来る．ここで

$$\boldsymbol{y} = \boldsymbol{Ux} \tag{6}$$

とおいた．分解後の求解ステップにおいては，まず$\boldsymbol{Ly} = \boldsymbol{b}$から$\boldsymbol{y}$を求め，次いで$\boldsymbol{Ux} = \boldsymbol{y}$から$\boldsymbol{x}$を求める．このとき，前者はインデックスが大きい方へ向かって，後者は小さい方へ向かって値が決定されることから，それぞれ前進代入・後退代入（合わせて前進後退代入）と呼ぶ．

4.3 反復法

適当な初期解$\boldsymbol{x}^{(0)}$を仮に定め，近似解の更新を行っていく方法は，更新アルゴリズムを反復させることから，反復法と呼ばれる．[1]k回の反復後の近似解$\boldsymbol{x}^{(k)}$による残差$\left\|\boldsymbol{b}-\boldsymbol{Ax}^{(k)}\right\|$の値が 0 に近くなれば，真の解$\hat{\boldsymbol{x}}$に収束したとして計算を止める．係数行列に 0 が多く疎である場合，LU 分解ではメモリ使用量が増えてしまうが，反復法では追加がなく計算機資源を節約出来るため，大規模計算にも向く．

4.4 直接法と反復法の比較

直接法と反復法にはそれぞれ利点・欠点が存在し，場合により使い分けることが求められる．反復法の利点として，特に疎行列を扱う場合，メモリを節約出来る点，前処理の利用によって直接法よりも速く解を求めることが可能になる．

一方の直接法の利点としては，反復法では収束しない形の線形方程式も扱えることを第一に挙げられる．物性値の大きく異なる複合材料や，係数行列の対角成分に 0 の入る接触解析を扱う場合，収束解を得られない．また，反復回数の定まらない反復法と異なり，予め計算回数を求められる．加えて，行列を分解するという操作は，右辺ベクトルである$\boldsymbol{b}$が変化しても結果が変わらないため，異なる試行による複数の$\boldsymbol{b}$ に対して再利用出来る．

5 疎行列線形方程式の数値解法

5.1 解法の全体像

求解は主に 5 ステップからなる．行列積への分解では成分の値が 0 であったところに非零の値が入ることがあり，これをフィルインと呼ぶ．初めのオーダリングでは，フィルインがなるべく生じないようにインデックスの並び替えを行う．シンボリック分解では，フィルインの個数を確定するとともに，次の分解操作における処理の依存関係を求める．マルチフロンタル法は，係数行列$\boldsymbol{A}$が疎な場合に並列化を実現する手法である．前進後退代入は一般の LU 分解の場合と同一である．最後には反復的な解の改良を行うが，これは得られた解における数値誤差を緩和するため，残差を利用して解を更新する操作である．[3]

5.2 オーダリング

行列のインデックスに適当な順序を付けて入れ替えることからオーダリングと呼ばれる．インデックスを頂点・非零成分を辺とするグラフを作成し，領域A,BをA の頂点とBの頂点を両端とする辺が 1 つもないようにとる．またA,B 以外の頂点集合を，セパレータSとする．ここで行列のインデックスを，A,B,S の順にそれぞれに含まれる頂点に対応するインデックスの順に並び替える．すると，A,B の設定より，非対角領域に零行列が現れる．[4)]

このようにして得られる行列を縁付きブロック対角行列と呼び，操作を再帰的に繰り返すことで，行列は再帰的に縁付きブロック対角となる．[4)]このとき，非対角部分が零行列となっている場合には独立に分解操作が行えるため，並列化に優れる．

5.3 シンボリック分解

フィルインの数を決定するのがシンボリック分解である．また，フィルインの場所が特定されることに伴って，成分ごとの依存関係も決定出来る．依存関係を木構造で表したものを消去木と呼び，次のマルチフロンタル法の演算においては，一番依存のない葉に当たる部分から順に分解操作を行う．

$\boldsymbol{A}$が対称行列であることを前提とする．インデックスの小さい方から順に走査し，着目しているiに依存するj のうち最小のものを消去木でのi の親とする．また，iに依存するがjには依存しないインデックスkがあったとき，kは j にも依存する形となり，フィルインが生じる．

5.4 マルチフロンタル法

有限要素法により離散化される線形方程式は大規模かつ疎な行列$\boldsymbol{A}$を左辺に持つ．すると反復法が望ましい手法であるが，収束しない可能性もあるため，確実な求解には直接法が向いている．疎性を生かし，かつ並列計算可能な直接法として，行列の依存のない部分単位で分解を行うマルチフロンタル法が存在する．

着目しているインデックスkについての分解は図2のアルゴリズムのように行われる．ここで，$\boldsymbol{F}^k$をインデックスkについてのフロンタル行列，最終的に得られる$\boldsymbol{U}^k$ をアップデート行列と呼ぶ．[4,5)]$\boldsymbol{F}^k$ははじめの段階では第k行（列）の非零成分のみからなる行列であり，これに消去木においてkの子たるi のアップデート行列$\boldsymbol{U}^i$を足し込む．この加算演算では，$\boldsymbol{F}^k$と$\boldsymbol{U}^i$のインデックスを揃えた上で値を足す必要があり，そのまま和をとる訳にいかないため，Extend-Add 演算と呼ばれる（図 2 中の⊕）．これを全ての子iについて行う．[5)]

5.5 前進後退代入

LU 分解と共に説明したものと手法は同一である．

$$\boldsymbol{y} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{b} \tag{7}$$

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{L}\boldsymbol{y} = \boldsymbol{b} \tag{8}$$

ベクトル$\boldsymbol{y}$を式(7)で定めると，式(1)・(3)より(8)が得られる．これを基に，まず$\boldsymbol{L}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{b}$から$\boldsymbol{y}$を，次いで$\boldsymbol{L}\boldsymbol{x} = \boldsymbol{y}$から$\boldsymbol{x}$を決定する．

5.6 反復的な解の改良

反復法においては，各ステップにおける近似解ベクトルを元の式に代入し，そこで得られた残差が十分小さくなるまで反復を繰り返した．一方直接法においては，数学的な操作のみに依拠して求解を行うため，数値誤差が生じる．これを補正するため，解ベクトル$\boldsymbol{x}$を元の式に代入し，反復法のように残差ベクトルを小さくする操作を行う．

直接法の求解過程での数値誤差を低減する物であるため，求解に利用した浮動小数点数よりも高い精度を利用することが求められる．倍精度で求解を行ったなら倍々精度や 4 倍精度を用いる．[3)]

1: **function** FACTOR(k)
2: $\boldsymbol{F}^k = \begin{bmatrix} a_{k,k} & a_{k,q_1} & a_{k,q_2} & \cdots & a_{k,q_s} \\ a_{k,q_1} & 0 & 0 & \cdots & 0 \\ a_{k,q_2} & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{k,q_s} & 0 & 0 & \cdots & 0 \end{bmatrix}$
3: **for** i is k's child **do**
4: FACTOR(i)
5: $\boldsymbol{F}^k = \boldsymbol{F}^k \oplus \boldsymbol{U}^i$
6: **end for**
7: **for** $i = 0$ to t **do**
8: $l_{q_i,k} = \boldsymbol{F}^k_{i,0} \,/\, \sqrt{\boldsymbol{F}^k_{0,0}}$
9: **end for**
10: **for** $j = 1$ to t **do**
11: **for** $i = j$ to t **do**
12: $\boldsymbol{U}^k_{i,j} = \boldsymbol{F}^k_{i,j} - l_{q_i,k} \times l_{q_j,k}$
13: **end for**
14: **end for**
15: **end function**

Figure 2 Algorithm of multifrontal method

6 並列計算

6.1 計算機の分類

1960 年代に提案された計算機の分類では，SISD, SIMD, MISD, MIMD の 4 類型が存在する．MIMD におけるアプリケーションの実装として，SPMD という考え方がある． SPMD では，複数のプロセッサにおいて同じプログラムを走らせるものの，プロセッサによって処理の内容を変化させ，また異なるデータを扱わせる．共有メモリ型・分散メモリ型という分類では，SPMD は各プロセスで異なる演算・データを扱うことから，分散メモリ型での実行に適している．

6.2 Message Passing Interface

SPMD を実現する規格として，Message Passing Interface (MPI) が存在する． このとき，あるプロセッサが持つデータに異なるプロセッサから直接アクセスすることが出来ない．そのため，MPI にはデータのやり取りに関する様々な関数が定義されている．[6)]

6.3 MPI によるプロセス管理インタフェース

MPI にはプロセス生成・管理機能が存在する．これは，初めから全てのプロセッサでプログラムを動かすのでなく，ある Master と呼ばれるプロセスが実行途上に Worker と呼ばれるプロセスの生成・消去を行うモデルである．異なる内容のプログラムを実行する Worker を生成することも可能なため，異なるハードウェアでの実行を組み合わせたり，データによって実行数や内容を変えたりすることが容易になる．

6.4 MPI_COMM_SPAWN によるプロセス管理

MPI においてプロセス生成を行う関数として，MPI_COMM_SPAWN が定義されている. [6]引数としてアプリケーション名をとることで，コマンドで SPMD の実行をするのと同じ働きをなす. また，プロセスの集団であるコミュニケータには，Master/Worker 双方の入るインターコミュニケータを指定することで, 両者での通信が可能となる．[6]

7 マルチフロンタル法の実装

7.1 疎行列の格納形式

成分に 0 の多い疎行列を利用する場合には，積の結果が 0 となるため不要な演算が多く, 2 次元配列を利用するのは非効率である．したがって，非零成分のみを格納する形式が利用される． 直感的に分かりやすいのは，全ての非零成分について行・列・値を記載した COO 形式である

一方で，非零成分の行のインデックスの代わりに，「ある行の非零成分の開始点を示すポインタ」を格納するのが CSR 形式である.

7.2 オーダリングの実装

既存のライブラリである METIS による Nested Dissection 法のルーチンを利用することで実装した．元の$\boldsymbol{A}$ と置換後の行列$\boldsymbol{A'}$の対応を示す置換ベクトルと逆演算のベクトルが返ってくるため, これを利用して置換行列$\boldsymbol{P}$を作成し，行列同時置換を行う.

7.3 シンボリック分解の実装

プログラムとしては，消去木の作成とフィルインの個数決定を同時に行う．第 0 行から順に非零成分を持つ列のうち最小のインデックスを記録していくことで, 親子関係を定められる. その上で，子に依存し親に依存しないインデックスに対してフィルインがあると決定する.

7.4 マルチフロンタル法の実装

図 2 のアルゴリズムの通り，分解操作は親から子へと再帰的に行われる．よって，消去木の根に当たる，最大のインデックスを初めの分解対象として与え, 以下再帰的に全てのインデックスに対して分解操作を行う. 並列化に当たっては, 依存のない消去木のサブツリーごとにプロセスを生成し分解を実行する.

7.5 前進後退代入の実装

実装内容は 4．2 のアルゴリズムに従った手順である．ここで, 入力に CSR 形式の疎行列を用いた場合について考えると, ある行の非零成分の個数とそのデータ領域は行の開始点を示すポインタにより分かっている．よって，これらと各非零成分の列に対応するベクトルの成分の積和を考えれば，行列ベクトル積が容易に計算出来る.

7.6 反復的な解の改良の実装

演算に用いた精度よりも高精度な型を用いるため，求解までを倍精度で行うとし，改良は QD というライブラリを利用して倍々精度で行う. ここでも行列ベクトル積と前進後退代入を利用することで，解ベクトル$\boldsymbol{x}$の更新を行える.

8 数値実験

8.1 計算条件

FrontISTR の計算時に出力した係数行列・右辺ベクトルのファイルを入力に利用した．行列の記法は CSR 形式である. また, 検算用に解ベクトルのファイルも利用した.

計算機としては, 手元のラップトップの他, クラスタ計算機として, 研究室の管理するサーバ, 本学情報基盤センターのスーパーコンピュータを用いた.

8.2 逐次計算プログラムの結果

逐次プログラムの実行結果と反復法による解ベクトルの値を比較すると，差の絶対値は10^{-16}オーダーであり，正しい解を得られたと言える.

異なる環境における実行時間の比較では，計算機の性能差はあるものの, いずれもマルチフロンタル法での分解に実行の大部分を要していた．既存ライブラリである MUMPS との差は大きく，効率化が求められる.

8.3 並列計算プログラムの結果

プロセス生成・消去を行うプログラムを作成し, まずプロセス生成を行わないよう設定したところ, 逐次プログラムと同等の結果を得られたことから, アルゴリズムとして破綻していないと言える.

ラップトップでの実行で 2 プロセス生成，さらにそこから 2 プロセスずつの生成 (4 並列) を指示したところ，正しく消去木上のノードを辿ることが確認された．ただしメモリ競合の関係か，最後まで実行はされなかった．クラスタにおける実行では，Master 部分は問題ないものの, Worker の生成が不安定であった. 同一計算ノード内を指定しても，通信エラーとなることがあり，異種環境に対応するには改善が必要である.

9 結論

行列の疎性を生かし, かつ並列な行列積への分解を行える線形方程式の直接解法として, マルチフロンタル法の実装を行った. 異種環境の利用も想定し, 動的にプロセス管理を行う並列化を組み合わせたところ, 逐次と同様の実行順を確認出来たが，正確な結果の取得には至らなかった. 今後の課題として, 動的プロセスによるアルゴリズムの完成に加え，Worker プロセスの対象を多様化出来る構成, また正定値対称でない$\boldsymbol{A}$ への対応を挙げられる．そのように求解可能な問題の幅を広げ, アプリケーションとして実装することにより, 実社会のものづくり現場に寄与することが今後の到達点である.

文献

1) 日本計算工学会・編：「線型方程式の反復解法」，（2017）.
2) O. C. Zienkiewicz, K. Morgan：”Finite Elements & Approximation”，（1983）.
3) 寒川光，藤野清次，長嶋利夫，高橋大介：「HPC プログラミング」，(2009).
4) 山本有作：計算工学，11（4），1458（2006）.
5) A. Gupta，G. Karypis，V. Kumar：IEEE Transactions on Parallel and Distributed Systems, 8（5），502（1997）.
6) The Open MPI Project https://www.open-mpi.org/