サイエンスへの招待

有限要素法シミュレーションの並列計算法

東京理科大学 理工学部 機械工学科 助教 遊佐（ゆさ） 泰紀（やすのり）

はじめに

機械の一生にはシミュレーション技術が深く関わっている。機械を設計・生産するときは，変形や応力が基準を満たすかどうかを調べるためにシミュレーションが用いられている。また，供用中の機械に対しては，補修や部品交換の必要性を判断するためにシミュレーションが活用されつつある。さらに，災害や事故などで機械が壊れると，原因究明と対策のためにシミュレーションが行われることがある。

変形や応力，つまり連続体の力学現象のシミュレーションには一般に有限要素法が用いられる。有限要素法では**図１**のように物体を複数の要素で分割する。図中の六面体一つ一つが要素であり，各六面体の頂点が節点である。要素で分割された物体はメッシュと呼ばれる。有限要素法では，各要素内で物理量の分布を適当な式で近似することで，支配方程式を近似的に解く。図１のメッシュの円筒の両端に１MPaの引張荷重を与えたときの有限要素法シミュレーションの結果を**図２**に示す。色は引張方向の応力を示す。赤色の部分は応力値が大きく，紫色の部分は応力値が小さい。また，分かりやすさのために変形を拡大して可視化した。中央の赤色の部分が大きく変形していることが分かる。このような単純形状の物体は瞬く間にシミュレーションできるが，現実の機械のシミュレーションでは必要な要素数・節点数が大きくなり，要素数・節点数次第では非常に長い計算時間を要する。例えば，筆者が所属する研究室では約２ヵ月間の計算時間を要する有限要素法シミュレーションを行ったことがある。ビジネスの現場では，２ヵ月間は文字通り「待つことができない」ほどに長い時間であろう。

計算時間を大幅に削減する最も有力な方法

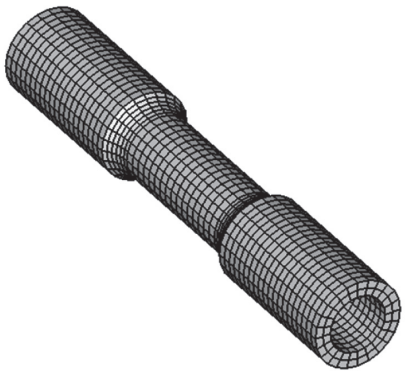

図１　物体の要素分割（メッシュ）の例

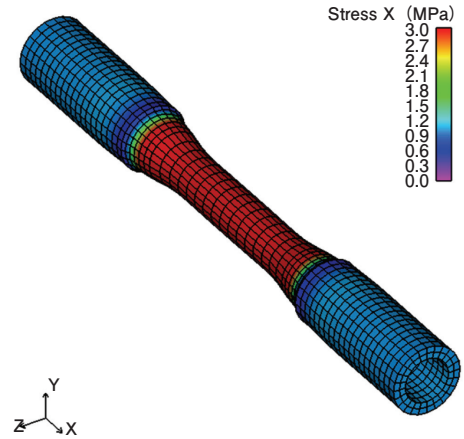

図２　図１のメッシュを用いた有限要素法シミュレーションの結果の例

の一つが並列計算である。現在，スーパーコンピュータ，ワークステーション，パーソナルコンピュータ，スマートフォンなど，世の中の計算機の多くは並列計算機である。しかも，CPU（Central Processing Unit）の複数のコアを利用することに留まらず，GPU（Graphics Processing Unit）などのアクセラレータを汎用計算に転用することもある。筆者が所属する研究室が所有する並列計算機の例を**写真１**と**写真２**に示す。写真１は２つのCPUと１つのGPUを有する計算機である。それぞれのCPUやGPUは複数のコアを有する。写真２は８台のノードを接続したクラスタである。これらのような並列計算機を活用することで，「待つことができる」ほどの計算時間の有限要素法シミュレーションを実現できるかもしれない。

本稿では，最近の有限要素法シミュレーションでよく用いられるいくつかの並列計算法を紹介し，それぞれの特徴を簡単に説明する。そして，実際に各並列計算法を用いて有限要素法シミュレーションを行い，計算時間を比較する。以上から，節点数があまり大きくないメッシュを高速にシミュレーションできる手法，それほど高速ではないが節点数がかなり大きくてもシミュレーションできる手法など，それぞれの並列計算法が適材適所であることを示す。

有限要素法と連立一次方程式

一般に，有限要素法のアルゴリズムの中で最も計算時間を要する部分は連立一次方程式の求解である。有限要素法では，要素ごとの力・変位関係を重ね合わせて全体の連立一次方程式を作成する。この連立一次方程式を求解すると節点上の変位が求まる。そして，変位からひずみや応力を計算する。有限要素法の連立一次方程式にはいくつかの特徴がある。第１に，多くの場合，連立一次方程式の係数行列は正定値対称行列である。したがって，正定値対称行列向けの連立一次方程式解法を用いることが合理的である。第２に，連立一次方程式の未知変数の数はおおよそ節点数の３倍の値である。３という数字はx，y，zの三方向の変位や力に対応する。一般に，未知変数の数が大きくなればなるほど長い計算時間を要する。第３に，係数行列は疎行列である。疎行列とは，成分のほとんどがゼロである行列のことである。もし疎行列であることを考慮しない場合，係数行列の成分ごとに８Bの倍精度浮動小数点数を用いるとして概算すると１万節点で6.7 GB，10万節点で670 GBものメモリ容量が必要になる。なお，現在一般的な計算機のメモリ容量は数GBから数十GB程度である。

以上の特徴を有する連立一次方程式の解法として，直接解法と反復解法のそれぞれを紹

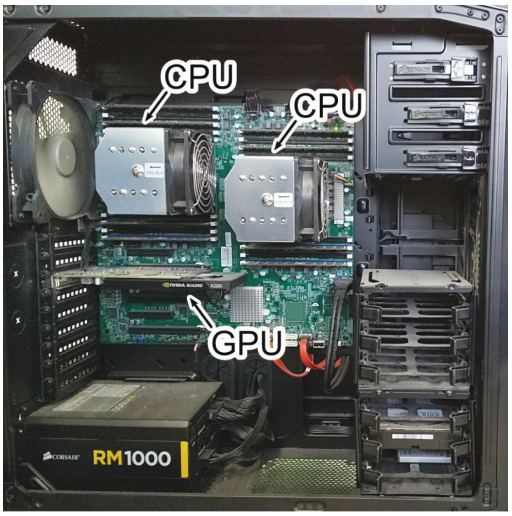

写真１　２つのCPUと１つのGPUを有する計算機


写真２　８台のノードを接続したクラスタ

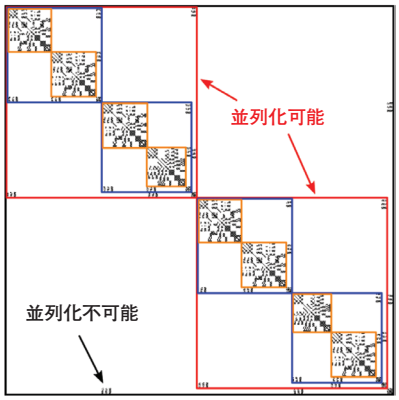

図３　ネステッドディセクション法で行番号・列番号を並び替えた行列の例

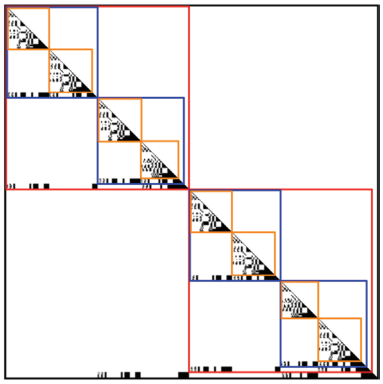

図４　図３の行列をLDL分解すると得られる行列$\boldsymbol{L}$

介する。直接解法の代表として，対称行列に対して広く用いられているLDL分解法を紹介する。特に，近年よく用いられる疎行列直接解法（スパースダイレクトソルバー）を対象とする。反復解法の代表として，正定値対称行列では最も一般的である共役勾配法を紹介する。また，直接解法と反復解法の両方に基づく領域分割法を紹介する。

疎行列直接解法

直接解法の一つであるLDL分解法では，連立一次方程式の係数行列$\boldsymbol{K}$を

$$\boldsymbol{K} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\mathrm{T}} \qquad (1)$$

のように３つの行列の積の形で分解する。ただし，$\boldsymbol{L}$は対角項より上の成分がすべてゼロの行列，$\boldsymbol{D}$は対角項以外の成分がすべてゼロの行列である。係数行列が$\boldsymbol{L}^{\mathrm{T}}$の連立一次方程式，係数行列が$\boldsymbol{D}$の連立一次方程式，係数行列が$\boldsymbol{L}$の連立一次方程式を順に解くと，当初の連立一次方程式の解が求まる。特に疎行列直接解法では，$\boldsymbol{K}$や$\boldsymbol{L}$を疎行列として取り扱うために省メモリである。なお，疎行列直接解法のプログラミングには線形代数だけでなくグラフ理論の高度な知識が必要であるため，公開されているソフトウェアライブラリを利用することが多い。PARDISO（https://www.pardiso-project.org/）やMUMPS（http://mumps.enseeiht.fr/）が有名な疎行列直接解法ライブラリである。

疎行列直接解法の並列化にはネステッドディセクション法が用いられることが多い。ネステッドディセクション法では，行列の行番号・列番号を並び替えることで，式(1)のLDL分解を並列に実行できるような係数行列を作成する手法である。これもソフトウェアライブラリの利用が一般的であり，METIS（http://glaros.dtc.umn.edu/gkhome/views/metis）やSCOTCH（https://www.labri.fr/perso/pelegrin/scotch/）が有名である。METISを用いてネステッドディセクション法で行番号・列番号を並び替えた行列の例を**図３**に示す。黒色が非ゼロの成分，白色がゼロの成分である。赤色で囲んだ２つのブロック行列を独立，つまり並列に処理できる。さらに，青色で囲んだ４つの部分で４並列，オレンジ色で囲んだ８つの部分で８並列となり，再帰的な並列性を有する。また，黒色の矢印で指した部分は並列化することができない成分である。図３の行列をLDL分解すると**図４**のような行列$\boldsymbol{L}$が得られる。行列$\boldsymbol{K}$ではゼロ（白色）であった成分の一部が行列$\boldsymbol{L}$では非ゼロ（黒色）になっていることが分かる。これはフィルインと呼ばれる。疎行列直接解法では，フィルインの分だけメモリを多く使用する必要がある。以上のように，疎行列直接解法は並列化の方法が複雑であり，また，並列化できない部分がそれなりに残るため，一般に並列化効率が

図５　行列・ベクトル積の並列化

あまり高くない。つまり，並列プロセス数を増やしてもそれほど高速にならない。

共役勾配法

共役勾配法は反復解法の一つであり，行列・ベクトル演算とベクトル・ベクトル演算の組み合わせを繰返し行うことで収束解を得る方法である。各反復ステップの中では，一般に行列・ベクトル積

$$\boldsymbol{q} = \boldsymbol{K}\boldsymbol{p} \qquad (2)$$

が最も計算時間を要する。ただし，$\boldsymbol{p}$および$\boldsymbol{q}$はベクトルである。この行列・ベクトル積を**図５**のように並列に計算する。並列プロセスの数だけ行列$\boldsymbol{K}$を短冊状に分割し，それぞれのブロックを各並列プロセスが担当する。このとき，式(2)の右辺のベクトル$\boldsymbol{p}$はすべての並列プロセスから参照されるが，計算結果の左辺のベクトル$\boldsymbol{q}$は並列プロセスごとに分散的に計算される。

同様にして，ベクトル・ベクトル演算の並列化の方法を説明する。ベクトルの和の並列化方法を**図６**に示す。各並列プロセスがベクトルの一部を担当し，和を計算する。この演算は並列プロセスごとに完全に分散的に行われるため，並列プロセス間の通信は必要ない。続いて，ベクトルのスカラー倍の並列化方法を**図７**に示す。灰色で表されているスカラーはすべての並列プロセスが同じ値を保持している。もしすべての並列プロセスがこのスカラー値を保持していない場合は，並列プロセス間で通信して値をコピーする。そして，並列プロセスごとに分散的にベクトルの

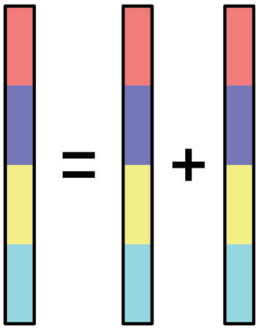

図６　ベクトルの和の並列化

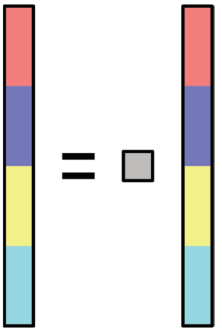

図７　ベクトルのスカラー倍の並列化

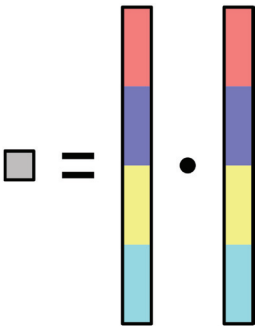

図８　ベクトルの内積の並列化

一部のスカラー倍を計算する。最後に，ベクトルの内積の並列化方法を**図８**に示す。各並列プロセスがベクトルの一部の内積を計算し，その後，並列プロセス間で通信することでベクトル全体の内積を計算する。

以上のように，共役勾配法は並列化の方法が単純であるため，一般に並列化効率が高めである。

また，図５のような行列の分割は**図９**のようなメッシュの領域分割に対応する。図９は図１のメッシュをMETISで領域分割したも

のである。４つの色がそれぞれの部分領域を示している。本質的には同様であるが，行列の代わりにメッシュを分割する実装方法が用いられることもある。

領域分割法

領域分割法は直接解法と反復解法の両方に基づく解法である。領域分割法では，図９の部分領域ごとに直接解法による有限要素法シミュレーションを繰返し行い，最終的に部分領域間の連続条件・平衡条件を満たす収束解を得る。この反復計算には，共役勾配法などの反復解法の枠組みを適用することができる。領域分割法では，部分領域ごとに独立した有限要素法シミュレーションを並列に行うため，一般に並列化効率が高めである。また，領域分割法では図９のような並列プロセスの数だけの領域分割ではなく，**図10**のような並列プロセス数以上の領域分割を行うことがある。このアプローチによって，各部分領域の要素数・節点数を直接解法が得意な範囲に設定することができる。

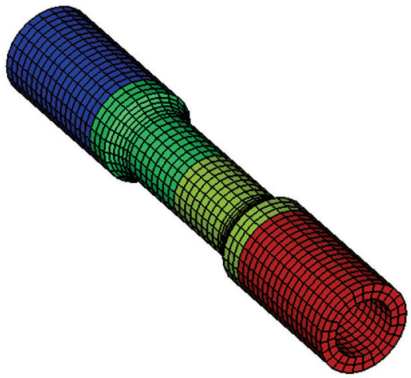

図９　図１のメッシュを４個の部分領域に分割した例

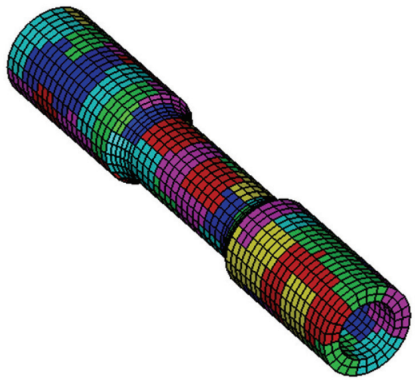

図10　図１のメッシュを60個の部分領域に分割した例

表１　計算時間の比較のために使用したメッシュの要素数と節点数

ID	要素数	節点数
1	8,448	10,680
2	26,112	32,880
3	67,584	76,464
4	208,896	235,872
5	540,672	576,096
6	1,671,168	1,778,880

各並列計算法の計算時間の比較

図１のメッシュの細かさを変えたものを６個用意し，それぞれのメッシュに対して各並列計算法の計算時間を計測した。各メッシュの要素数と節点数を**表１**に示す。疎行列直接解法には，MUMPSライブラリに基づく自作プログラムを用いた。共役勾配法と領域分割法には，オープンソース有限要素法シミュレーションソフトウェアADVENTURE_Solid（https://adventure.sys.t.u-tokyo.ac.jp/）を用いた。Intel Core i5-4590 CPUとDDR3-1600メモリ16 GBを搭載する計算機を用い，CPUの４つのコアに対してMPI（Message Passing Interface）による４プロセスの並列計算を行った。

図11に計測した計算時間を示す。横軸が節点数，縦軸が計算時間である。黒線が疎行列直接解法，赤線が共役勾配法，青線が領域分割法を示す。図11の横軸および縦軸の値域を変えたものを**図12**に示す。ID 3とID 3より節点数が小さいメッシュでは疎行列直接解法が一番高速である。しかし，ID 4のメッシュでは疎行列直接解法が一番低速である。さらに，ID 5とID 5より節点数が大きいメッシュでは，メモリ使用量が計算機のメモリ容量を超過したため，疎行列直接解法ではシミュレーションすることができなかった。一般に，

直接解法では要素数・節点数が大きくなると，計算時間とメモリ使用量が急峻に増大することが知られている。また，共役勾配法と領域分割法を比較すると，共役勾配法の方が総じて高速であるが，計算時間の変化の傾向はよく似ていることが分かる。以上に示した計算時間は，並列計算法の詳細なアルゴリズムやプログラミング，計算機によって大きく変わることに注意する。

おわりに

最近の有限要素法シミュレーションでよく用いられる並列計算法として，疎行列直接解法，共役勾配法，領域分割法の３つを紹介し，各並列計算法の計算時間を比較した。疎行列直接解法は節点数が比較的小さい問題を高速にシミュレーションできるが，節点数が大きくなると必要なメモリ容量が非常に大きくなり，シミュレーションできなくなってしまう。一方，共役勾配法と領域分割法は節点数が大きくなっても問題なくシミュレーションできる。以上のように，節点数に応じて最適な並列計算法が異なることを示した。

本稿では，特に共役勾配法や領域分割法については簡単のために最も単純な手法のみを紹介した。しかし，最新の研究や実用的なソフトウェアでは，前処理法という手法を用いることで反復計算のさらなる高速化を図っていることが多い。前処理法の具体的な手法は多岐にわたっている。興味のある読者は科学技術計算フレームワークのPETSc（https://www.mcs.anl.gov/petsc/）やTrilinos（https://trilinos.org/）の動向を参照されたい。また，本稿では並列計算機の詳細を省略したが，単に並列計算機と言ってもノードの数，ノードあたりのCPUの数，CPUあたりのコアの数，コアあたりのSIMD（Single Instruction, Multiple Data）のway数など，さまざまなレベルの並列性がある。それぞれに対するプログラミングの方法は異なるものであり，また，各並列計算法の向き不向きもある。

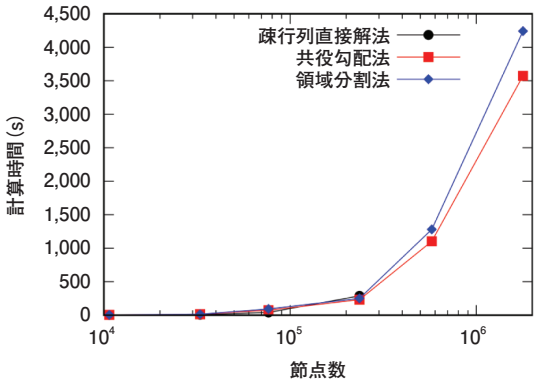

図11　節点数に対する計算時間の変化

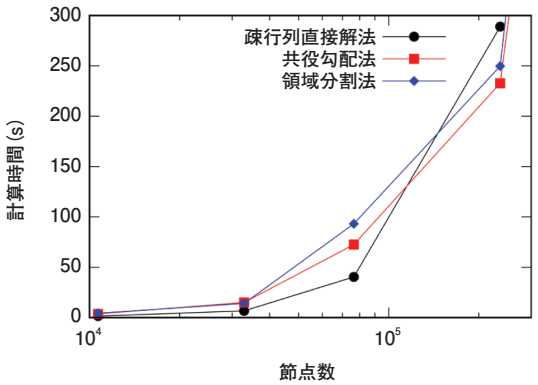

図12　図11の横軸および縦軸の値域を変えたもの

2020年度から施行される新学習指導要領で注目されているプログラミング教育の目的は，必ずしもプログラミングそのもののスキルを身に付けることではなく，プログラミング的思考力というものを育むことであるらしい。一方，本記事で対象とした有限要素法シミュレーションは市販ソフトウェアで行われることが一般的であるため，どうしてもブラックボックス的に見られがちであるように筆者は感じている。並列計算についても，ソフトウェアのオプションをオンにすると「よく分からないまま」に並列計算をしてくれるため，中身を意識する機会はあまりないかもしれない。本稿によって読者が有限要素法などのシミュレーションの中身を意識し，プログラミング的思考力を養う一助となれば幸いである。