チュートリアル

疎行列の連立一次方程式の解を求めるための直接法のアルゴリズムを概説し、オープンソースソフトウェアや有償ソフトウェアとして利用できる疎行列の直接法ソルバについて基本的な使用方法を説明します。なお、チュートリアル記事は1ページ目のみを本誌に掲載し、続きは日本計算工学会HP上で公開していますので、そちらも併せてご参照ください。

疎行列直接法ソルバ入門

緒方 隆盛

1　はじめに

大規模計算機シミュレーションにおいて、疎行列ソルバ（疎行列を係数とする方程式の解を求めるソフトウェア）は重要な機能の一つです。例えば、CAE解析でよく用いられる有限要素法は、物理現象から導出される偏微分方程式を線形化・離散化することで、大規模な疎行列を係数とする連立一次方程式を解くことに帰着されます。

本チュートリアルは、疎行列ソルバの中でも疎行列の連立一次方程式$\boldsymbol{Ax = b}$の解$\boldsymbol{x}$を直接法で求めるソルバ（以降、疎行列直接法ソルバと記載）に焦点を当てて説明します。線形代数の数値解法になじみがない方、疎行列直接法ソルバを利用したことがない方、ソルバをブラックボックスとして使用しているアプリケーション開発者を対象に、疎行列直接法ソルバの代表的なアルゴリズムを概説します。また、以下の疎行列直接法ソルバの基本的な使用方法について、簡単な例を交えて説明します。

- HeteroSolver
- Parallel Direct Sparse Solver for Clusters Interface
- MUMPS

これから自作のプログラムに疎行列ソルバを組み込んでみたい方や、アプリケーション内の計算時間を短縮するために疎行列ソルバを変えて評価してみたいと考えている方にとって、一助となれば幸いです。

筆者紹介

おがた りゅうせい
2003年、日本電気株式会社入社。ベクトル型スーパーコンピュータSXシリーズの科学技術計算ライブラリASL, MathKeisanの開発および製品サポートや、アプリケーションの高速化支援などを担当。現在は、主にSX-Aurora TSUBASA向けの科学技術計算ライブラリNEC Numeric Library Collectionの開発および製品サポートに従事。

2　疎行列直接法ソルバと反復法ソルバ

疎行列ソルバを大別すると、「直接法ソルバ」と「反復法ソルバ」があります。

直接法ソルバは、所定の計算回数で疎行列を三角行列の積に分解した後に解を求める手法です。一般的に知られている直接法ソルバの利点は、反復法ソルバでは収束しにくい悪条件の問題（疎行列の条件数が大きい問題）でも解が得られる場合があることです。欠点としては、メモリ使用量が反復法ソルバに比べて大きく、計算機のリソース不足になる場合があることです。

一方、反復法ソルバは、適当な初期解から繰り返し近似解を求め、最終的な近似解を求める手法です。反復法ソルバの利点としては、疎行列の性質と反復アルゴリズムや前処理の適性によっては少ない反復回数で高速に解が得られ、メモリ使用量が少ないという点が挙げられます。その反面、繰り返しの過程で生じる計算誤差が蓄積し、解が収束しにくい場合や、正しい解に収束しない場合があることなどが欠点として挙げられます。

実際には、多くの疎行列直接法ソルバでは、得られた解の精度を改良するために収束計算を繰り返します。また、反復法ソルバの前処理として直接法ソルバを用いて近似解を求めることで解の収束を加速することもあります。このようなアルゴリズムの高度化・複雑化により、疎行列ソルバのアルゴリズムに詳しくないユーザにとっては、ソルバで提供している関数やサブルーチンが何のために提供されていて、何をしているのかわかりにくいと感じるかもしれません。

次項では疎行列直接法ソルバの大まかな流れを知っていただくために、疎行列直接法ソルバで行っている処理について概説します。

続きはWebで
日本計算工学会誌「計算工学（Vol.25, No.2）」
HP: https://www.jsces.org/activity/journal/

3　疎行列直接法ソルバ概説

一般的な疎行列直接法ソルバは、解析フェーズ、数値分解フェーズ、求解フェーズの3つのフェーズに分かれています。各フェーズを順番に処理すると、連立一次方程式$\boldsymbol{Ax} = \boldsymbol{b}$の解$\boldsymbol{x}$が得られます（$A$：行列、$\boldsymbol{x},\boldsymbol{b}$：ベクトル）。

3.1　解析フェーズ

解析フェーズは、ユーザが入力した疎行列の非ゼロ要素の位置（行と列）や値を解析するフェーズです。後続のフェーズの演算量とメモリ使用量を削減するための解析や変換処理を行い、後続のフェーズの演算量やメモリ使用量を見積もります。後続のフェーズのアルゴリズムによっては、正確な演算量やメモリ使用量を算出することもできます。解析フェーズで使用するアルゴリズムやパラメータの違いは、ソルバ全体の計算量やメモリ使用量に大きく影響します。

1.　リオーダリング

疎行列を三角行列に分解すると、入力の疎行列で値がゼロである部分にゼロ以外の値が出現します。この要素はフィルインと呼ばれています。フィルインは計算量やメモリ使用量に大きく影響します。リオーダリングはフィルインがなるべく発生しないように、入力された疎行列の行番号や列番号を並び替える処理です。リオーダリングにより、数値分解のために必要な計算量やメモリ使用量の増大を抑えることができます。

代表的なリオーダリングのアルゴリズムとしては、AMD法（Approximately Minimum Degree[1]）やND法（Nested Dissection[2,3]）があります。AMD法は、疎行列の隣接情報をグラフとして表したときに、分解後の隣接節点数が近似的に最小になるように並び替える手法です。AMD法は疎行列で構成されるグラフを局所的に見てフィルインを減らすため、“ボトムアップアプローチ”と表現されます。一方、ND法は、隣接情報のグラフを大域的に見て分割し、行または列の番号を付け替える方法です。ND法は、“トップダウンアプローチ”と表現されます。グラフ分割ライブラリとして著名なMETIS[4]はND法をベースにしています。

多くの直接法ソルバではリオーダリングのアルゴリズムを選択できるようにしており、アルゴリズムの設定を変更すると、計算時間の短縮やメモリ使用量の削減できることがあります。

2.　消去木生成

リオーダリングにより行と列を入れ替えた後、疎行列を三角行列に分解する処理の計算順序を決めます。計算順序の依存関係は、消去木と呼ばれる行番号または列番号を節点とする木構造グラフとして表されます。図1は、対称疎行列Aについて計算する順番を木構造グラフで表した図です。図1右のグラフの節点の番号は、対称疎行列Aの行または列番号です。後続の数値分解フェーズでは、図1右の木構造グラフの下から上に向かって、行列要素の消去処理を進めます。

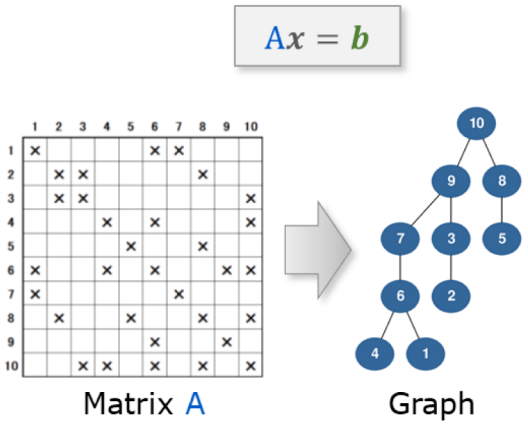

図1　疎行列と消去木

3.　シンボリック分解

消去木が求まると、疎行列について後続の数値分解フェーズで三角行列に分解する場合に、どの位置（行と列）にフィルインが発生するかを予め解析することができます。フィルインの位置を求める処理がシンボリック分解[8]です。この処理により、数値分解フェーズで必要な計算量やメモリ使用量の概算値が得られます。

4.　その他

解析フェーズでは、上述した処理以外にも、計算誤差を軽減するために疎行列の対角要素の絶対値を大きくする処理や[5]、計算の依存関係を共通化できる処理をまとめて効率よく演算するためのスーパーノード検出[6]や、後続のフェーズで使用するメモリの確保などが行われます。

3.2　数値分解フェーズ

数値分解フェーズでは、解析フェーズで得られた結果を使って、入力疎行列Aを三角行列または対角行列の積に分解します。Aが非対称行列の場合はLU分解やLDU分解、対称行列やエルミート行列の場合はLDL^t分解やLL^t分解がおこなわれます。多くの場合、数値分解フェーズが全フェーズの中で最も計算時間がかかるため、多くの直接法ソルバではターゲットとするアーキテクチャ向けに高度に最適化されたBLAS[7]を使い性能を向上させています。

3.3　求解フェーズ

求解フェーズでは、数値分解フェーズで得られた行列を使って前進・後退代入し、最終的な解を求めます。例えば、数値分解フェーズでLU分解した場合は、連立一次方程式$\boldsymbol{Ax} = \boldsymbol{b}$を式(1)のように変換することで解ベクトル$\boldsymbol{x}$を求められます。

$$
\begin{aligned}
&\boldsymbol{Ax = b}\\
&\boldsymbol{(P^tAP)P^tx = P^tb}\\
&\boldsymbol{LUP^tx = P^tb}\\
&\boldsymbol{x = PU^{-1}L^{-1}P^tb}
\end{aligned}
\tag{1}
$$

ここで、Pはリオーダリングの置換行列、Lは下三角行列、Uは上三角行列です。また、解ベクトルxの精度が十分ではないと判断した場合は反復改良により、解の精度を改善させます。解の反復改良は、ソルバによっては求解フェーズとは独立したフェーズとして実装しているものもあります。

4　疎行列格納形式

疎行列直接法ソルバで連立一次方程式を解くためには、疎行列データを配列として表す必要があります。疎行列データを配列に格納する形式は、「疎行列格納形式」と呼ばれます。疎行列格納形式の代表的な形式として、以下の形式が挙げられます。

- CSR形式
- CSC形式
- COO形式
- ELLPACK形式
- JDS形式
- BSR形式

この他にもメモリ上に配置した疎行列データへのアクセス性能を改善するための格納形式が多数ありますが、ここでは後述で使用するCSR形式とCOO形式について説明します。

CSR形式は、行列の非ゼロ要素を行方向に詰め、行列の1行目から順番に番号付けする格納形式です。CSR形式は、疎行列を3つの配列で表します。

- 非ゼロ要素の値を表す配列　　（図2の*aval*）
- 非ゼロ要素の列番号を表す配列（図2の*iaind*）
- 各行の先頭の番号を表す配列　（図2の*iaptr*）

例として、以下の疎行列Aを CSR形式で表すとします。

$$
A = \begin{pmatrix}
a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0\\
0 & a_{22} & a_{23} & 0 & 0 & 0 & 0\\
0 & 0 & a_{33} & a_{34} & a_{35} & 0 & a_{37}\\
a_{41} & 0 & 0 & a_{44} & 0 & 0 & 0\\
0 & 0 & a_{53} & 0 & a_{55} & a_{56} & 0\\
0 & 0 & 0 & 0 & a_{65} & a_{66} & a_{67}\\
0 & 0 & a_{73} & 0 & 0 & a_{76} & a_{77}
\end{pmatrix}
\tag{2}
$$

式(2)の疎行列Aについて CSR形式で表すと、3つの配列（*aval, iaind, iaptr*）には以下のように格納します。ここで、*aval*は浮動小数点データ型の配列*iaind*と*iaptr*は整数型の配列です。

COO形式は非零要素の値、行番号、列番号のそれぞれを配列にした形式です。

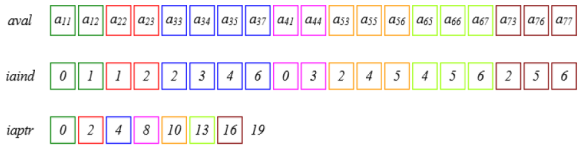

図2　CSR形式の例

- 非ゼロ要素の値を表す配列　　（図3の*aval*）
- 非ゼロ要素の列番号を表す配列（図3の*iarow*）
- 非ゼロ要素の行番号を表す配列（図3の*iacol*）

式(2)の行列AをCOO形式で格納する場合の例を図3に示します。

図3　COO形式の例

COO形式はCSR形式と異なり、全ての非ゼロ要素に対応する行番号を配列（*iarow*）に保持します。このため、同一の疎行列を格納する場合、CSR形式よりも、COO形式の方がメモリ使用量は大きくなります。

なお、図2および図3の例ではC言語用に行列の行番号、列番号、先頭の位置を0から開始していますが、Fortran用に1から開始することもあります。

5　基本的な直接法ソルバの使用方法

今回は、3つの疎行列直接法ソルバを例に、C言語の分散メモリ並列プログラムから直接法ソルバの使用する方法を説明します。

5.1　HeteroSolver

HeteroSolver[8]は、NEC製スーパーコンピュータSX-Aurora TSUBASA向けに開発された疎行列直接法ソルバであり、NEC Software Development Kit for Vector Engineに含まれるNEC Numeric Library Collectionのライブラリの一つとして提供されています。

図4にSX-Aurora TSUBASAのアーキテクチャの概要を示します。SX-Aurora TSUBASAはx86プロセッサ（VH）部と、アプリケーションの演算処理を行うベクトルエンジン（VE）部により構成され、PCI Expressで接続されています。

HeteroSolverの特長は、SX-Aurora TSUBASAのヘテロジニアス環境を有効活用するために、疎行列直接法の解析フェーズをx86プロセッサ側処理し、数値分解フェーズと求解フェーズを演算性能が高いベクトルエンジン側で処理することです。

CSR形式で格納された非対称行列の連立一次方程式の解を求める場合について、HeteroSolverの使用例を図5に示します。ここで、非対称行列の非ゼロ要素の値（CSR形式の配列*aval*）はdouble型で与えられているものとします。

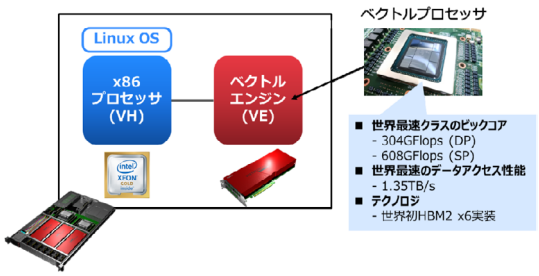

図4　SX-Aurora TSUBASA のアーキテクチャ概要

HeteroSolverのAPIは、HS_handle_t型のハンドル変数を介して、3. で述べた解析フェーズ、数値分解フェーズ、求解フェーズを処理します。図5の例では、入力疎行列Aと右辺ベクトル$\boldsymbol{b}$をMPIのランク0で設定し、求解フェーズの終了後、解ベクトルは求解フェーズの関数PHS_solve_rd()の第7引数（配列x）に格納されます。

```
#include <mpi.h>
#include <heterosolver.h>
<中略>
// MPIの初期化
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

// 入力データの設定
if (myrank == 0) {
  aval   = …  // 非ゼロ要素の値(CSR形式)
  iaind  = …  // 非ゼロ要素の列番号(CSR形式)
  iaptr  = …  // 各行の先頭位置(CSR形式)
  b      = …  // 右辺ベクトル
}

// ハンドルの初期化
HS_handle_t hnd;
PHS_init_handle(&hnd, N, N, HS_UNSYMMETRIC,
HS_DCSR, HS_MASTER, 1, N, MPI_COMM_WORLD);

//  解析フェーズ
PHS_preprocess_rd(hnd, iaptr, iaind, aval);

//  数値分解フェーズ
PHS_factorize_rd(hnd, iaptr, iaind, aval);

//  求解フェーズ
PHS_solve_rd(hnd, iaptr, iaind, aval, nrhs, b,
x, &res);

// ハンドルの終了
PHS_finalize_handle(hnd);

// MPIの終了処理
MPI_Finalize();
```

図5　HeteroSolverの使用例（C言語用）

HeteroSolverでは、複数のMPIランクでの疎行列データの入出力や、リオーダリングのアルゴリズム変更、解の反復改良の反復回数の指定などの細かな設定は、ハンドルのオプションとして設定できます。ハンドルのオプションを設定するための関数として関数PHS_set_option()が提供されています。オプションの詳細はHeteroSolverのユーザーズガイド[8]に記載されています。

```
// ハンドルへのオプション設定
PHS_set_option(hnd, …, );
```

図6　HeteroSolverのオプション設定

5.2　Parallel Direct Sparse Solver for Clusters Interface

Parallel Direct Sparse Solver for Clusters Interface[9]は、Intel MKLに含まれる疎行列直接法ソルバです。

Parallel Direct Sparse Solver for Clusters Interfaceを使用して、CSR形式で格納された非対称行列の連立一次方程式の解を求める場合の例を図7に示します。

```
#include <mpi.h>
#include <mkl.h>
#include <mkl_cluster_sparse_solver.h>
<中略>
// MPIの初期化
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

// MPIコミュニケータの変換 (C→Fortran)
int comm =  MPI_Comm_c2f( MPI_COMM_WORLD );

// パラメータの初期設定
void *pt[64] = { 0 };
MKL_INT iparm[64] = { 0 };
MKL_INT maxfct, mnum, phase, msglvl, error;

// パラメータの初期設定
MKL_INT mtype = 11    // 疎行列Aが非対称行列

//C言語用にCSR形式の配列の開始番号を0に設定
iparm[34] =  1;

// 入力データの設定
if (myrank == 0) {
  aval   = …  // 非ゼロ要素の値(CSR形式)
  iaind  = …  // 非ゼロ要素の列番号(CSR形式)
  iaptr  = …  // 各行の先頭位置(CSR形式)
  b      = …  // 右辺ベクトル
}
```

```
//  解析フェーズ（前ページの続き）
phase = 11;
cluster_sparse_solver
 (pt, &maxfct, &mnum, &mtype, &phase, &n,
 aval, iaptr, iaind, &idum, &nrhs, iparm,
 &msglvl, b, x, &comm, &error);

//  数値分解フェーズ
phase = 22;
cluster_sparse_solver
 (pt, &maxfct, &mnum, &mtype, &phase, &n,
 aval, iaptr, iaind, &idum, &nrhs, iparm,
 &msglvl, b, x, &comm, &error);

//  求解フェーズ
phase = 33;
cluster_sparse_solver
 (pt, &maxfct, &mnum, &mtype, &phase, &n,
 aval, iaptr, iaind, &idum, &nrhs, iparm,
 &msglvl,b, x, &comm, &error);

// メモリ解放
phase = -1;
cluster_sparse_solver
 (pt, &maxfct, &mnum, &mtype, &phase, &n,
 aval, iaptr, iaind, &idum, &nrhs, iparm,
 &msglvl,b, x, &comm, &error);

// MPIの終了処理
MPI_Finalize();
```

図7　Parallel Direct Sparse Solver for Clusters Interfaceの使用例（C言語用）

Parallel Direct Sparse Solver for Clusters Interfaceでは、解析フェーズ、数値分解フェーズ、求解フェーズの関数名は共通のcluster_sparse_solver()となっています。各フェーズの違いは、5番目の引数phaseに整数値を指定して区別します。図7では、解析フェーズでphase=11、数値分解フェーズでphase=22、求解フェーズでphase=33に指定していますが、各フェーズをまとめて処理することもできます。例えば、phase=12の場合は解析フェーズと数値分解フェーズを処理でき、phase=13の場合は解析フェーズから求解フェーズまでをまとめて処理できます。

また、HeteroSolverと同様に、複数のMPIランクでの疎行列データの入出力や、リオーダリングのアルゴリズム変更、解の反復改良の反復回数の変更などの細かな設定も可能です。設定する際は、MKL_INT型の配列iparamに整数値を設定します。

5.3　MUMPS

MUMPS[10]はCeCILL-C ライセンスで公開されているオープンソースの疎行列直接法ソルバです。ソースコードの大部分はFortranで作成されていますが、C言語から利用するためのCインタフェースも提供されています。

疎行列の格納形式はCOO形式が利用できます。MUMPSを使うためには、MUMPS独自の構造体[DSZC]MUMPS_STRUC_Cを定義し、その構造体のメンバに入力データの性質（対称性、格納形式）やソルバの詳細を指定するための制御パラメータを設定します。ここで、[DSZC]は精度を表す接頭辞です。（D：倍精度実数、S：単精度実数、Z：倍精度複素数、C：単精度複素数）

図8は、COO形式で格納された非対称行列の連立一次方程式の解をMUMPS（version 5.2.1）で求める場合の使用例です。非対称行列の非ゼロ要素の値（COO形式の配列aval）はdouble型で与えられているものとします。

```
#include <mpi.h>
// 倍精度実数用インクルードファイル
#include <dmumps_c.h>
<中略>

// 倍精度実数用の構造体
DMUMPS_STRUC_C id;

// MPIの初期化
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

// MPIコミュニケータの変換（C→Fortran）
int comm =  MPI_Comm_c2f( MPI_COMM_WORLD );

// 構造体の初期化
id.comm_fortran = comm;
id.sym  = 0;      // 疎行列Aが非対称行列
id.job  = -1;     // 初期化処理を指定
dmumps_c(&id);    // 初期化処理

// 制御パラメータの設定
//   エラーや警告メッセージ等の出力を抑止
id.icntl[0]=0; id.icntl[1]=0;
id.icntl[2]=0; id.icntl[3]=0;

<…各種制御パラメータの設定…>

// 入力する疎行列Aと右辺ベクトルbをセット
if (myrank == 0) {
  id.n   = n;     // 疎行列Aの行数(=列数)
  id.nnz = nnz;   // 疎行列Aの非ゼロ要素数
  id.a   = aval;  // COO形式のaval
  id.irn = iarow; // COO形式のiarow
  id.jcn = iacol; // COO形式のiacol
  id.rhs = b;     // 右辺ベクトル
}
```

```
//  解析フェーズ
id.job=1;          // 解析フェーズのフラグ
dmumps_c(&id);     // 解析フェーズを実行

//  数値分解フェーズ
id.job=2;          // 数値分解フェーズを指定
dmumps_c(&id);     // 数値分解フェーズを実行

//  求解フェーズ
id.job=3;          // 求解フェーズを指定
dmumps_c(&id);     // 求解フェーズを実行

// 構造体DMUMPS_STRUC_Cの終了処理
id.job=-2;         // 終了処理を指定
dmumps_c(&id);     // 終了処理

// MPIの終了処理
MPI_Finalize();
```

図8　MUMPSの使用例（C言語、MPI並列）

制御パラメータは構造体のメンバ配列icntlを使って設定することができ、入力データの与え方やリオーダリングの種類、解の改良の反復回数などを指定できます。また、3. で述べた分析フェーズ、数値分解フェーズ、求解フェーズは構造体のメンバ変数jobに整数値で設定して実行します。求解フェーズが終われば、連立一次方程式の解は、右辺ベクトルとして与えた配列id.rhsに上書きされます。なお、MUMPSのCインタフェースを使用する場合は、疎行列の行番号や列番号はFortran用に1から開始する添え字番号を入力することに、注意する必要があります。

図8の例では疎行列の格納形式としてCOO形式を用いましたが、有限要素法のプログラム向けに剛性行列を要素ごとに格納する形式にも対応しています。また、MUMPSはメモリ不足の場合にファイルに入出力する機能も提供しています。

6　さいごに

今回は、疎行列用の連立一次直接法ソルバを使う上で必要となる代表的なアルゴリズムと疎行列の格納形式について概説し、3つの疎行列直接法ソルバの使用方法を概説しました。

このチュートリアルでは詳しくご紹介できませんでしたが、紹介した疎行列直接法ソルバではオプションや設定パラメータを変更することで、ソルバをユーザプログラムにカスタマイズして使うことができます。また、設定を変更するとで、計算量やメモリ使用量を削減できることもあります。既定値を変更して試してみることをお勧めします。

■参考文献

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff, An approximate minimum degree ordering algorithm, SIAM J. Matrix Anal. Appl., 17, pp. 886-905, (1996)

[2] A. George, Nested dissection of a regular finite element mesh, SIAM J. Numer. Anal., 10, pp. 345-363, (1973)

[3] A. George, An automatic one-way dissection algorithm for irregular finite-element problems, SIAM J. Numer. Anal., 17, pp. 740-750, (1980)

[4] http://glaros.dtc.umn.edu/gkhome/metis/metis/overview/

[5] X. S.Li, and J. W. Demmel,A Scalable Sparse Direct Solver Using Static Pivoting, In Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, (1999)

[6] T. A. Davis: Direct methods for sparse linear systems, pp. 37-67, SIAM (ISBN 978-0-898716-13-9), (2006)

[7] http://www.netlib.org/blas/

[8] https://www.hpc.nec/documents/sdk/SDK_NLC/UsersGuide/heterosolver/c/ja/index.html

[9] https://software.intel.com/en-us/mkl-developer-reference-c-intel-mkl-pardiso-parallel-direct-sparse-solver-interface

[10] http://mumps.enseeiht.fr/