SILC: Simple Interface for Library Collections

バージョン: 1.3 (2007年10月31日)

はじめに

本パッケージには, 行列計算ライブラリインタフェース SILC のソースコード, サンプルプログラム, およびドキュメントが含まれています.

SILC は, 行列計算ライブラリを計算環境やプログラミング言語に依存しない方法で利用できるようにする新しいフレームワークです. 従来の関数呼び出しによるライブラリの利用法では, ユーザプログラムが特定のライブラリに依存するため, 別のライブラリや計算環境を利用するにはソースコードの大幅な書き換えが必要です. 一方, SILC のユーザプログラムでは, 行列やベクトルなどのデータに名前を付けてサーバに預け, 計算は文字列 (数式) で指示します. 数式中の演算子は適当なライブラリ関数の呼び出しに翻訳されてサーバ側のメモリ空間で実行されます. そのため, 用いるライブラリや計算環境をユーザプログラムの修正なしに容易に変更できます. また, 計算指示に数式を用いるので, どのプログラミング言語でも同じ方法でライブラリを利用できます.

SILC の設計および実現方法については以下の論文をご参照下さい.

- 長谷川, 須田, 額田, 梶山, 中島, 高橋, 小武守, 藤井, 西田. 計算環境に依存しない行列計算ライブラリインタフェースSILC, 情報処理学会研究報告, 2004-HPC-100, pp.37-42, 2004年.

  http://www.ssisc.org/HPC100/hasegawa.pdf

- 梶山, 額田, 須田, 長谷川, 西田. 共有メモリ並列環境におけるSILCの実現と利用, 第34回数値解析シンポジウム講演予稿集, pp.49-52, 2005年.

  http://www.ssisc.org/NAS2005/kajiyama.pdf

SILC を動かしてみるには

SILC は Unix/Linux 環境および Windows 環境で動作します. Unix/Linux 環境ではソースコードをコンパイルする必要があります. Windows 環境ではコンパイル済みのバイナリパッケージが利用できます. バイナリパッケージの詳細については README.win32.ja ファイルをご覧下さい.

Unix/Linux 環境および Microsoft Windows 環境で SILC のソースコードをコンパイルして実行するには以下のソフトウェアが必要です.

- C/Fortran コンパイラ
- GNU Make
- GNU Bison
- GNU Flex

また, 以下の行列計算ライブラリを SILC サーバに組み込んで利用できます.

- BLAS, LAPACK (http://www.netlib.org/)
- Lis 1.0.2 (http://www.ssisc.org/lis/)
- FFTSS (http://www.ssisc.org/fftss/)

SILC のソースコード (`silc-1.3.tar.gz`) は以下の場所から入手できます:

http://www.ssisc.org/silc/index-ja.html

`silc-1.3.tar.gz` からソースコードを取り出すには次のコマンドを実行します:

```
gzip -cd silc-1.3.tar.gz | tar xvf -
```

これによりカレントディレクトリの `silc-1.3/src/` ディレクトリ内に SILC のソースコードが展開されます. このディレクトリに移動して下さい:

```
cd silc-1.3/src
```

次にコンパイルする環境に合わせて `make.inc` というファイルを作成します. `inc/` ディレクトリ内にいくつかの計算環境向けの作成例があります. これらの例を参考にしてコンパイラオプション, ライブラリファイルの場所等を定義して下さい.

GNU/Linux 環境で GCC を用いる場合は `inc/make.gcc` が利用できます. 次のようにコピーして `make.inc` を作成して下さい:

```
cp inc/make.gcc make.inc
```

Windows 環境では MinGW (http://www.mingw.org/) に含まれる GCC と GNU Make, および GnuWin32 (http://gnuwin32.sourceforge.net/) の GNU Bison と GNU Flex が利用できます. 以下のバージョンでの動作を確認しています:

- http://downloads.sourceforge.net/mingw/MSYS-1.0.10.exe
- http://downloads.sourceforge.net/mingw/MinGW-3.1.0-1.exe
- http://downloads.sourceforge.net/gnuwin32/bison-2.1.exe
- http://downloads.sourceforge.net/gnuwin32/flex-2.5.4a-1.exe

上記のソフトウェアをインストールした上で, `inc/make.mingw` をコピーして `make.inc` を作成して下さい:

```
cp inc/make.mingw make.inc
```

他の環境では `make.inc` ファイルを新規作成する必要があります. 詳しくは SILC 利用者マニュアル の2.1 節「SILC サーバのコンパイル」をご覧下さい.

`make.inc` ファイルを作成したら, make コマンドを実行して SILC サーバとサンプルプログラムをコンパイルします:

```
make
```

サンプルプログラムの実行方法は以下の通りです. まず SILC サーバを起動します (以下の例は Unix/Linux 環境の場合です. Windows 環境では `/` [スラッシュ] を `¥` [円マーク] に読み替えて下さい):

```
cd src/server
./server
```

次にサンプルプログラム (例: `src/client/demo3.c`) を実行します:

```
cd src/client
./demo3
```

このプログラムは三重対角行列を係数とする連立一次方程式を解きます. ||b-Ax|| = 3.784025e-12 のような出力が得られれば正常に動作しています (実際の値は計算環境によって変わります).

また, console プログラム (`src/client/console.c`) を用いて対話的に計算を行なうことができます:

```
./console
```

2×2行列 A とベクトル b の積 `A * b` および `b' * A` を求める例を以下に示します (ここで `'` [シングルクォート] は転置を表します):

```
> A = {1, 2; 3, 4}
> b = {5, 6}
> x = A * b
> pprint x
column vector, 2 elements of int
  [1] = 23
  [2] = 34
> x = b' * A
> pprint x
row vector, 2 elements of int
  [1] = 17
  [2] = 39
>
```

各行の `>` 以降がユーザの入力です. 計算を終了するには, Unix/Linux 環境では Ctrl-D を, Windows 環境では Ctrl-Z を入力します. console プログラムの詳細については README.console.ja ファイルをご覧下さい.

ユーザプログラムの作り方

SILC の応用プログラム (以下, ユーザプログラム) の作成方法は SILC 利用者マニュアル に記載されています. 利用者マニュアルでは, C 言語および Fortran で SILC のユーザプログラムを作成する方法を説明しています. また, Microsoft Windows 上でのユーザプログラムの開発については README.win32.ja ファイルを合わせてご参照下さい.

ユーザプログラムの開発には, C と Fortran のほか, オブジェクト指向スクリプト言語 Python (http://www.python.org/) が利用できます. ただし, Python によるユーザプログラムの開発方法を述べたドキュメントはまだ用意されていません. Python によるユーザプログラムの記述方法は C の場合とほぼ同じです. また, サンプルプログラムが `src/client/python/` ディレクトリにあります.

サンプルプログラム

SILC のユーザプログラムのサンプルが `src/client/` ディレクトリ以下にあります. 主なユーザプログラムを以下に示します.

- `demo3_ssi_cg.c`
- `demo3.c`
- `fortran/fdemo3.f`

  三重対角行列を係数とする連立一次方程式を解くユーザプログラムです. `demo3_ssi_cg.c` は, 求解に用いるライブラリ関数 `ssi_cg()` を直接 (SILC を介さずに) 呼び出すプログラムの例です. このライブラリ関数は連立一次方程式を共役勾配法 (CG 法) で解く関数です. 一方, `demo3.c` と `fortran/fdemo3.f` は SILC を利用して同じ連立一次方程式を解くC言語および Fortran のコード例です. また, `demo3.c` の Python 版が `python/demo3.py` にあります.

- `silc_cg.c`

  SILC の命令記述言語で実現した CG 法のプログラムです. `demo3.c` と同じ連立一次方程式を例題として与えています.

- `mm_crs.c`
- `mm_band.c`

  Matrix Market (http://math.nist.gov/MatrixMarket/) の行列データファイル (`.mtx` ファイル) を読み込んで連立一次方程式を解くプログラムです. `mm_crs.c` は行列を CRS 形式で SILC サーバに預けて反復解法 (CG 法など) で求解します. `mm_band.c` は行列を LAPACK の banded storage 形式でサーバに送り, 直接解法 (LU 分解法など) で解きます. また, コマンドラインオプションで行列の格納形式を変更できます.

- `console.c`

  SILC の命令記述言語で書かれた命令文を標準入力から読み込み, SILC サーバに送って実行する対話型のユーザプログラムです. サーバを電卓のように利用できます. 計算結果を見るには「pprint 変数名」と入力します. このプログラムの詳細については README.console.ja ファイルをご覧下さい.

未実装の機能

本ソフトウェアは開発途上のため未実装の機能があります. 主な未実装の機能を以下に示します.

- CRS 形式の疎行列に対する制約代入 (左辺に添字をともなう代入文)
- 要素ごと乗算および除算
- 3次元配列

回帰テスト

SILC の動作を検証するために回帰テスト (regression tests) が用意されています. 回帰テストはそれ自体が Python で書かれた SILC のユーザプログラムです. 実行には Python 2.4 以降が必要です. 回帰テストは以下の手順で実行します:

```
cd src/client/python
```

```
python test_dense.py
python test_sparse_crs.py
python test_leq.py
python test_augmented.py
python test_subscript.py
python test_concat.py
python test_put_matrix.py
python test_matrix_format.py
```

文献の引用についてのお願い

SILC を利用した研究成果について今後お書きになる研究論文のリファレンスに以下の文献を含めてくださるようお願いいたします.

- 長谷川秀彦, 須田礼仁, 額田彰, 梶山民人, 中島研吾, 高橋大介, 小武守恒, 藤井昭宏, 西田晃. 計算環境に依存しない行列計算ライブラリインタフェースSILC, 情報処理学会研究報告, 2004-HPC-100, pp.37-42, 2004.
- 西田晃. SSI: 大規模シミュレーション向け基盤ソフトウェアの概要, 情報処理学会研究報告, 2004-HPC-098, pp.25-30, 2004.

また, 本ソフトウェアを利用した研究成果 (研究論文, 応用ソフトウェアなど) を下記までお送りいただけると幸いです.

〒113-8656 東京大学大学院 情報理工学系研究科
西田 晃
E-mail: devel **at** ssisc.org

著作権および利用条件

本ソフトウェアの著作権は SSI プロジェクトにあります. あなたは所定の利用条件にしたがって自由に本ソフトウェアを利用できます. 本ソフトウェアは無保証です. 本ソフトウェアの詳細な利用条件については `LICENSE` ファイルをご覧下さい.

また、本パッケージには以下の著作物が含まれています. これらの著作物は当該著作物の利用条件にしたがって利用して下さい.

- `src/server/dgefa.f`, `src/server/dgesl.f`

  LINPACK: The Netlib (http://www.netlib.org/linpack/) より.

- `src/lib/mt/mt19937ar.c`, `src/lib/mt/mt19937ar.h`

  疑似乱数生成器 Mersenne Twister: 原本は以下の場所から入手できます.

  http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html

連絡先

バグ報告, ご要望, ご意見等をお寄せ下さい. 下記の宛先までお気軽にどうぞ.

SSI プロジェクト <devel **at** ssisc.org>

更新履歴

- バージョン1.3 (2007年10月31日)
  - src/client/console.c: SILC の命令記述言語の拡張仕様を実装した. 拡張仕様の詳細については README.console.ja ファイルを参照.
  - ベクトル, CRS 形式の疎行列の要素ごと乗算および要素ごと除算を実装.
  - 以下の組込み関数を追加した.
    - coremodule: ベクトルの各要素の平方根を計算する関数 sqrt(v)
    - sparse_crs: 疎行列を生成する関数 sparse(i, j, v, m, n)
    - sparse_crs: 対角行列を生成する関数 diag(v), diag(v, k)
    - sparse_crs: 対角要素を取り出す関数 diagvec(m)
  - 以下のモジュールを追加した.
    - leq_smsamg: Super Matrix Solver AMG version 3 (株式会社ヴァイナス) を呼び出すための演算モジュール (experimental).
    - leq_mp: 多倍精度反復解法ライブラリ mp_crs を呼び出すための演算モジュール (experimental).
  - 倍精度実数の無限大を表す定数 inf を定義した.
  - leq_lis モジュールに COCG 法の実装を追加した (experimental).
  - coremodule の関数 norm2 が複素ベクトルについて不正な値を返すバグを修正した.
  - CRS 形式の長方行列の転置の結果がおかしくなるバグを修正した.
  - CRS 形式の疎行列の加減算のバグを修正した.
  - 関数および手続きの引数の修飾子 (符合反転, 転置, 複素共役, 添字) が無視される問題を修正した.
  - SILC_PUT と SILC_EXEC を交互に行なうと変数の値がおかしくなる問題を修正した.
  - src/client/mm_crs.c: 残差ノルムの計算がおかしかった問題を修正した.
  - その他, 多数の軽微なバグ修正および改良を施した.
- バージョン1.2 (2006年11月12日)
  - 以下のモジュールを追加した.
    - leq_lis: 反復解法ライブラリ Lis を呼び出すための演算モジュール (http://www.ssisc.org/lis/ 参照).
    - fftss: 高速フーリエ変換ライブラリ FFTSS を呼び出すための演算モジュール (http://www.ssisc.org/fftss/ 参照).
    - sparse_jds: Jagged Diagonal Storage (JDS) 形式の疎行列をサポートする格納形式モジュール (experimental).
  - PUT/GET リクエストのデータ転送速度を改善した.
  - EXEC リクエストの文字数の上限を4096バイトに増やした.
  - EXEC リクエストに構文エラーが見つかった場合, SILC_EXEC がエラーを返すようにした.
  - CRS 形式の疎行列の減算, 行列ベクトル積, 行列積のバグを修正した.
  - CRS 形式の疎行列の転置演算を改良した.
  - いくつかの組込み関数と手続きを追加した.
    - split: 密行列版に加えて CRS 版を実装.
    - ones: 全要素が1の行列の生成.
    - srand, rand: 疑似乱数の初期化と乱数行列の作成 (Mersenne Twister).
    - svd: LAPACK を用いて特異値分解を計算.
  - スカラーリテラルの精度を指定できるようにした. これに合わせて虚数単位を表す定数 i の精度を倍精度から単精度に変更した.

    - Microsoft Windows (MinGW) 用の Makefile.mingw を追加した.
    - その他, いくつかのバグを修正した.
- バージョン1.1 (2005年11月25日)
    - 英語版の README.en ファイルと「SILC 利用者マニュアル」を追加した.
    - 共有ライブラリの動的リンク機能がない計算環境 (NEC SX-6i など) をサポートした.
    - NEC SX-6i 用の Makefile.sx を追加した.
    - その他, いくつかのバグを修正した.
- バージョン1.0 (2005年9月20日)
    - 最初の一般公開版.

$Id: README.ja,v 1.15 2007/10/31 05:18:48 kajiyama Exp $

Win32 用 SILC バイナリパッケージ

バージョン: 1.3 (2007年10月31日)

はじめに

本ファイルでは Win32 環境向けに作成された SILC バイナリパッケージの利用方法を説明します. バイナリパッケージは `silc-1.3-mingw32.zip` というファイル名の ZIP 形式のアーカイブファイルの形で配布されています. このパッケージには, 実行可能形式の SILC サーバ, サンプルプログラム, その他のバイナリファイルが含まれています (Windows XP SP2 での動作を確認しています). Windows マシンが手元にあり, もっとも手間のかからない方法で SILC を試してみたい方は, 本バイナリパッケージをご利用下さい.

Quick start

1. バイナリパッケージを展開してすべてのファイルを取り出します. 以下の説明では取り出したファイルが `C:¥silc-1.3¥` ディレクトリにあると仮定します.

2. コマンドプロンプトを開いて (例: スタートメニューから「すべてのプログラム」→「アクセサリ」→「コマンドプロンプト」を選択), 次の要領で `C:¥silc-1.3¥src¥server¥` をカレントディレクトリにします (各行の `>` 以降がユーザの入力です):

   ```
   > C:
   > cd ¥silc-1.3¥src¥server
   ```

   このディレクトリにある `server.exe` を実行して SILC サーバを起動します:

   ```
   > server
   ```

   サーバはフォアグラウンドで起動して以下のようなデバッグ情報を表示します:

   ```
   single thread
   load_modules("./modules/formats")
   silc_register_format("SILC:dense (column major)")
   silc_register_module("dense")
   silc_register_format("SILC:Band")
   silc_register_module("sparse_band")
   silc_register_format("SILC:CRS")
   silc_register_module("sparse_crs")
   silc_register_format("SILC:JDS")
   silc_register_module("sparse_jds")
   load_modules("./modules")
   silc_register_module("blasmodule")
   silc_register_module("coremodule")
   silc_register_module("leq_cg")
   silc_register_module("leq_gs")
   silc_register_module("leq_lis")
   silc_register_module("leq_lu")
   silc_register_module("linpackmodule")
   ```

3. 別のコマンドプロンプトを開いて `C:¥silc-1.3¥src¥client¥` ディレクトリをカレントディレクトリにします. このディレクトリには SILC のサンプルプログラムのソースファイルと実行可能形式ファイルが入っています:

```
> C:
> cd ¥silc-1.3¥src¥client
```

続いて SILC のユーザプログラム (例えば `demo3.exe`) を実行します:

```
> demo3
```

このプログラムは, もう一方のコマンドプロンプトで動作している SILC サーバに接続し, サーバの提供するライブラリルーチンのひとつを用いて連立一次方程式 Ax = b を解きます. このプログラムもデバッグ情報を表示しながら処理を進めます. 最後に, 以下の例のように実行時間 (単位は秒) と得られた解の残差ノルム ||b－Ax|| を表示して終了します:

```
0.063935s
||b-Ax|| = 3.839510e-015
```

実際の値は異なることがありますが, 同様の数値が得られていればプログラムは正しく動作しています.

4. 同じディレクトリにある console プログラム (`console.exe`) を用いて対話的に計算を行なうことができます:

```
> console
```

2×2行列 A とベクトル b の積 `A * b` および `b' * A` を求める例を以下に示します (ここで `'` は転置を表します):

```
> A = {1, 2; 3, 4}
> b = {5, 6}
> x = A * b
> pprint x
column vector, 2 elements of int
  [1] = 23
  [2] = 34
> x = b' * A
> pprint x
row vector, 2 elements of int
  [1] = 17
  [2] = 39
>
```

console プログラムを終了するには Ctrl-Z を入力します. このプログラムの詳細については README.console.ja ファイルをご覧下さい.

注意:.dll ファイルに関するエラーメッセージが表示されるときは以下のように環境変数 PATH を設定して下さい:

```
> set PATH=C:¥silc-1.3¥src¥server;%PATH%
```

ユーザプログラムの作り方

SILC の応用プログラム (以下, ユーザプログラム) の作成方法は SILC 利用者マニュアル に記載されています. 利用者マニュアルでは, C 言語および Fortran で SILC のユーザプログラムを作成する方法を説明しています.

ユーザプログラムの開発には, C と Fortran のほか, オブジェクト指向スクリプト言語 Python (http://www.python.org/) が利用できます. ただし, Python によるユーザプログラムの開発方法を述べたドキュメントはまだ用意されていません. Python によるユーザプログラムの記述方法は C の場合とほぼ同様です. また, サンプルプログラムが `src¥client¥python¥` ディレクトリにあります.

C または Fortran で SILC のユーザプログラムを作るには以下のことが必要です.

1. プログラムのソースコードの冒頭でヘッダファイルをインクルードします. C で作成する場合は `src¥client¥client.h` を, Fortran で作成する場合は `src¥client¥fortran¥client.h` を用います.
2. `src¥client¥client.c` または `src¥client¥libsilc.a` を WinSock2 ライブラリと共にユーザプログラムにリンクします.

例として, Microsoft Visual Studio .NET 2003 を用いる場合と MinGW (Windows 用の GCC) を用いる場合について, 行列 A の自乗 (行列積) を計算する C のプログラム `mmul.c` をコンパイルして実行するまでの手順を示します. プログラムの内容は以下の通りです (プログラム中の関数や定数については「利用者マニュアル」を参照して下さい):

```
#include <stdio.h>

#include "client.h"

int main(int argc, char *argv[])
{
  silc_envelope_t object;
  double A[4] = {1.0, 2.0, 3.0, 4.0};
  double X[4];

  SILC_INIT();

  object.v = A;
  object.type = SILC_MATRIX_TYPE;
  object.format = SILC_FORMAT_DENSE;
  object.precision = SILC_DOUBLE;
  object.m = object.n = 2;
  SILC_PUT("A", &object);

  SILC_EXEC("X = A * A");

  object.v = X;
  SILC_GET(&object, "X");

  SILC_FINALIZE();

  printf("A =¥n");
  printf(" %e %e¥n", A[0], A[2]);
  printf(" %e %e¥n", A[1], A[3]);
  printf("A * A =¥n");
  printf(" %e %e¥n", X[0], X[2]);
  printf(" %e %e¥n", X[1], X[3]);
}
```

- Microsoft Visual Studio .NET 2003 を用いる場合

  要点は以下の2点です.

  - 次のヘッダファイルと C ファイルをプロジェクトに追加する.
    - `src¥client¥client.h`

    - src¥client¥client.c
  - 以下のプロパティを設定する.
    - 〈構成プロパティ〉→〈リンカ〉→〈入力〉を選んで〈追加の依存ファイル〉にWinSock2 ライブラリ ws2_32.lib を追加する.
    - 〈構成プロパティ〉→〈C/C++〉→〈プリコンパイル済みヘッダー〉を選択して〈プリコンパイル済みヘッダーの作成/使用〉を〈プリコンパイル済みヘッダーを使用しない〉に設定する.

手順の詳細を以下に示します.

1. Microsoft Visual Studio .NET 2003 を起動して〈Visual C++ プロジェクト〉→〈Win32 コンソールプロジェクト〉を選択し, 新しいプロジェクトを作ります. プロジェクト名は mmul とします.

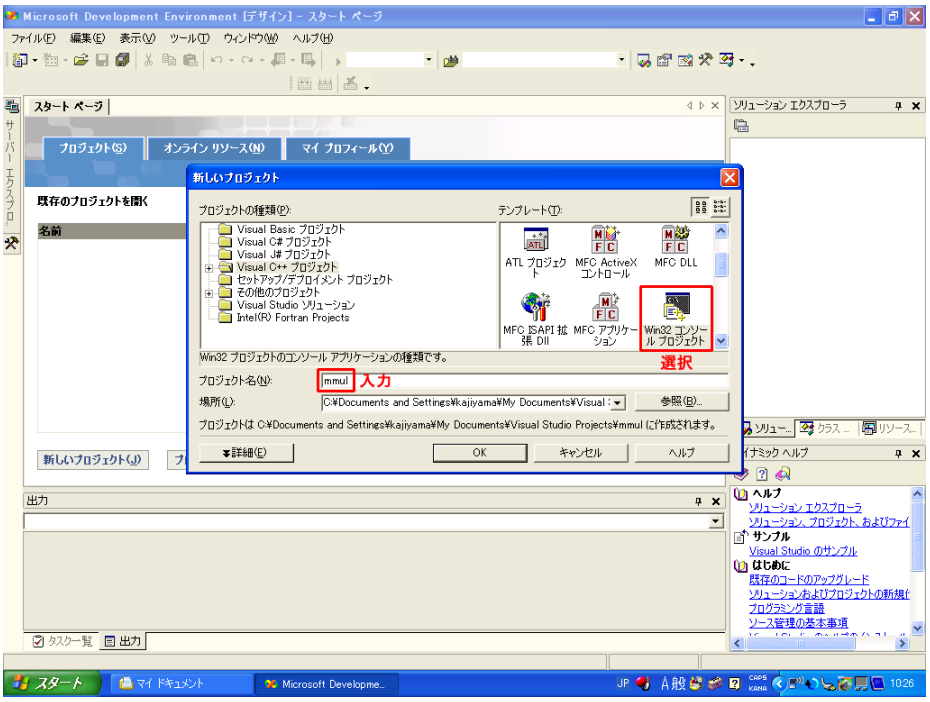

図1: プロジェクトの作成

2. プロジェクト作成時に以下のファイルが自動生成されます. これらは不要なのでフォルダごと削除します.

    - ソースファイル¥mmul.cpp
    - ソースファイル¥stdafx.cpp
    - ヘッダファイル¥stdafx.h
    - ReadMe.txt

    ただし, この操作ではファイルへの参照のみが削除されます. ファイルの実体は次のステップで削除できます.

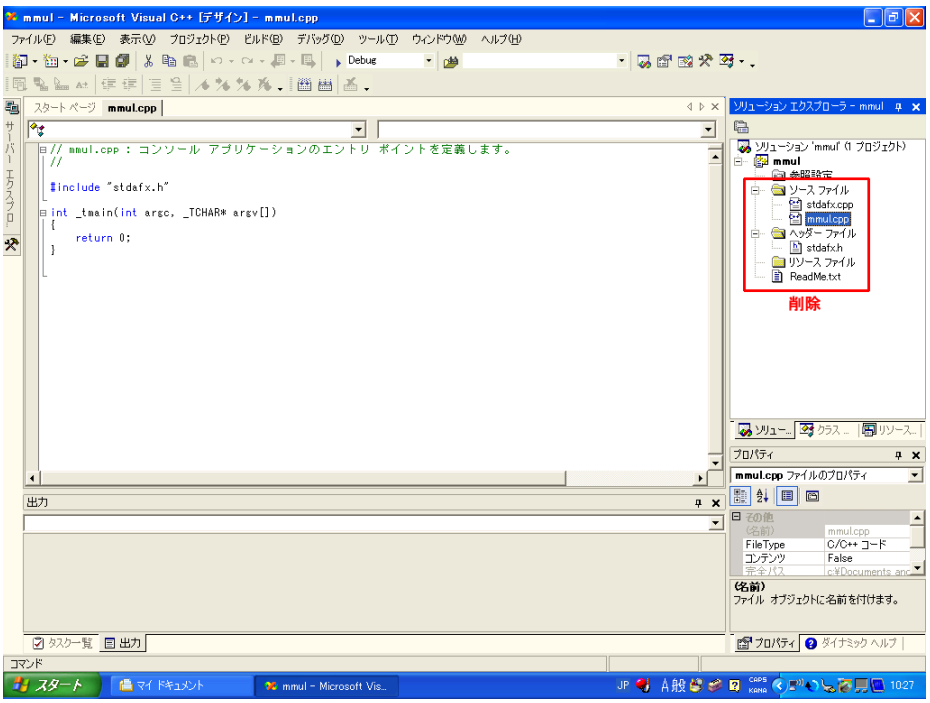

図2: 不要なファイルの削除

3. 以下のファイルを〈マイドキュメント〉配下の `Visual Studio Project¥mmul¥` ディレクトリに移動またはコピーします.

   - `mmul.c`
   - `src¥client¥client.c`
   - `src¥client¥client.h`

   また, ステップ 2. で挙げた不要なファイルも削除しておきます.

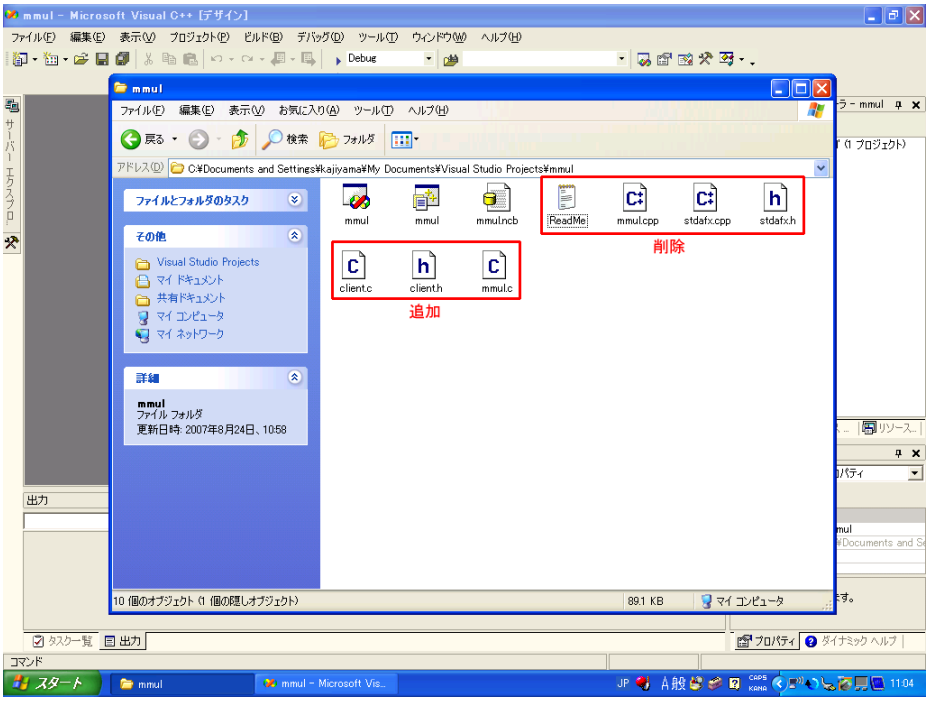

図3: ファイルの追加と削除

4. 〈プロジェクト〉→〈既存項目の追加〉を選択して, ステップ 3. で用意した3つのファイルをプロジェクトに追加します.

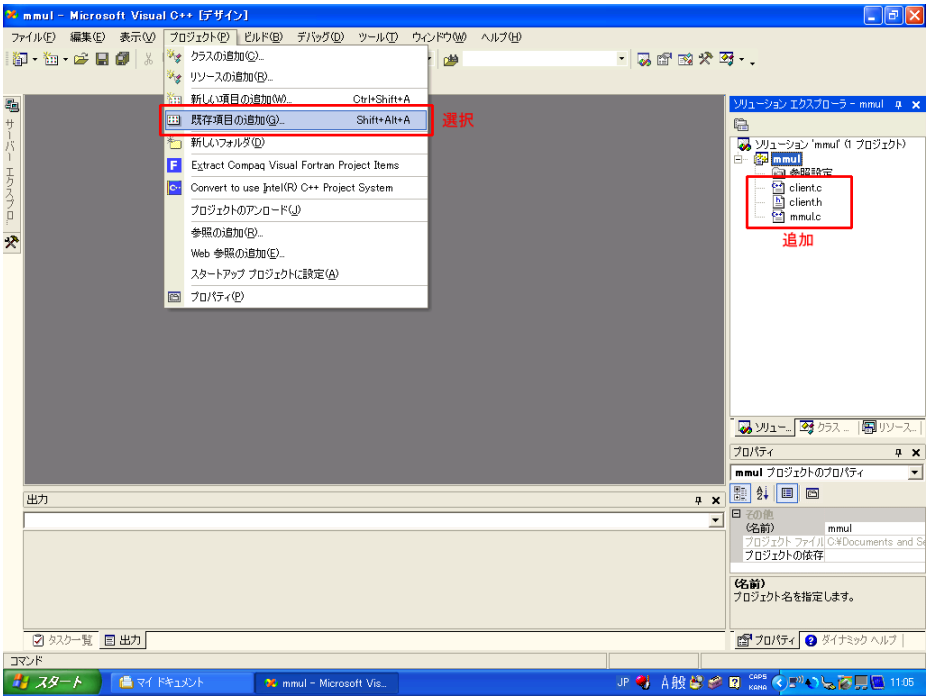

図4: 既存項目の追加

5. 〈プロジェクト〉→〈プロパティ〉を選択 (または図のボタンをクリック) して以下のプロパティを設定します.

   - 〈構成プロパティ〉→〈リンカ〉→〈入力〉を選択して〈追加の依存ファイル〉に WinSock2 ライブラリ `ws2_32.lib` を追加する.
   - 〈構成プロパティ〉→〈C/C++〉→〈プリコンパイル済みヘッダー〉を選択して〈プリコンパイル済みヘッダーの作成/使用〉を〈プリコンパイル済みヘッダーを使用しない〉に設定する.

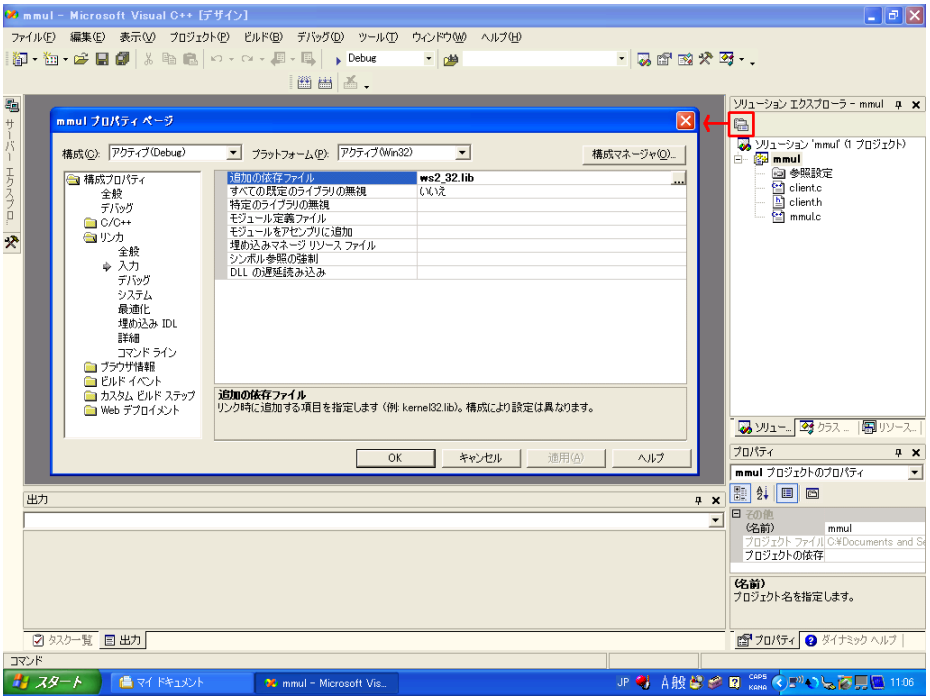

図5: プロジェクトのプロパティの設定

6. 〈ビルド〉→〈ソリューションのビルド〉を選択してコンパイルとリンクを行います. 正常終了すれば〈マイドキュメント〉配下の `Visual Studio Projects¥mmul¥Debug¥` ディレクトリに実行可能形式ファイル `mmul.exe` が作成されます.

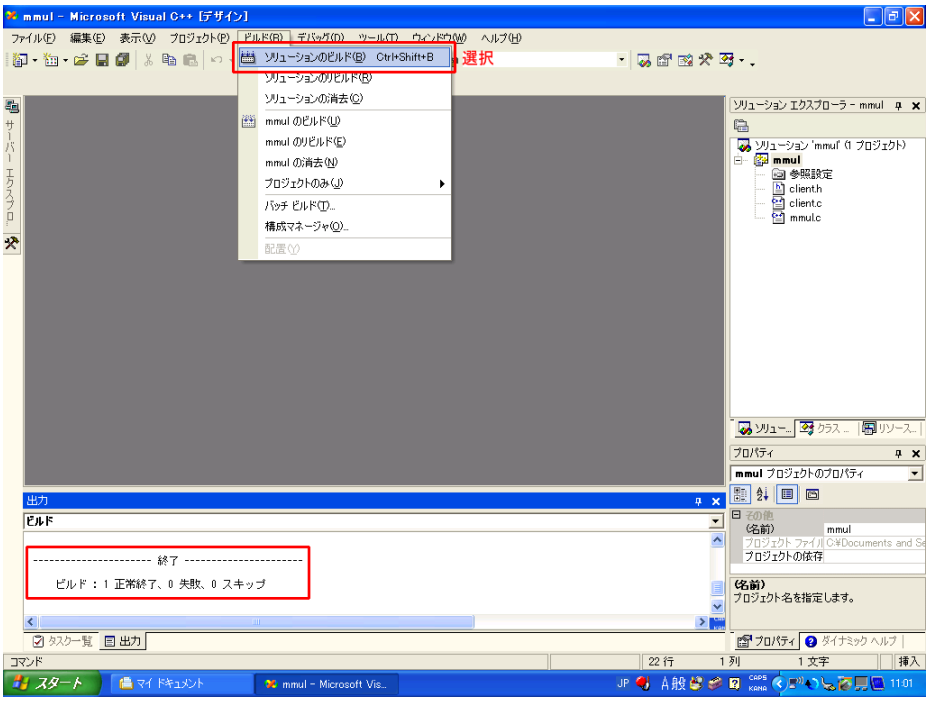

図6: プロジェクトのビルド

7. スタートメニューから〈アクセサリ〉→〈コマンドプロンプト〉を選択してコマンドプロンプトを開き, SILC サーバを起動します:

```
> C:
> cd ¥silc-1.3¥src¥server
> server
```

さらに別のコマンドプロンプトを開いてユーザプログラムを実行します:

```
> C:
> cd "My Documents¥Visual Studio Projects¥mmul¥Debug"
> mmul
connected to kajiyama on port 1639
number of formats = 4
  0: "SILC:dense (column major)"
  1: "SILC:Band"
  2: "SILC:CRS"
  3: "SILC:JDS"
Request: >A
Response: 200 OK
Request: :9:X = A * A
Response: 200 OK
Request: <X
Response: 200 OK
A =
 1.000000e+000 3.000000e+000
 2.000000e+000 4.000000e+000
A * A =
 7.000000e+000 1.500000e+001
 1.000000e+001 2.200000e+001
```

```
>
```

上記のような実行結果が得られれば正常に動作しています.

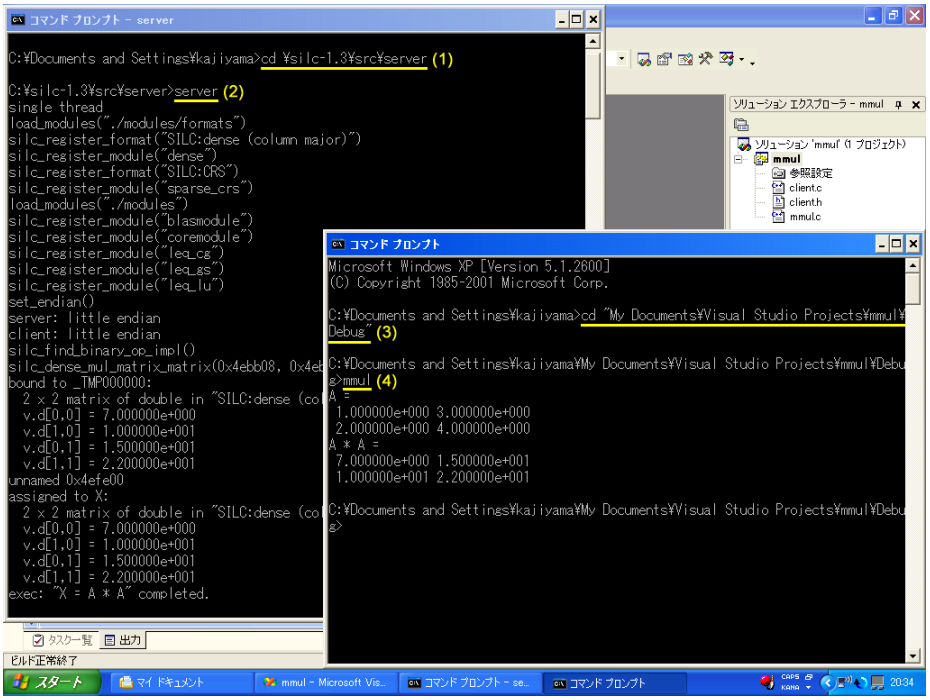

図7: SILC サーバとユーザプログラムの実行

- MinGW (GCC 3.2.3) を用いる場合

  MinGW (http://www.mingw.org/) は Windows 用の GCC です. 次の2つのインストーラを順に実行するだけで GCC 3.2.3 一式をインストールできます.

  - http://downloads.sourceforge.net/mingw/MSYS-1.0.10.exe
  - http://downloads.sourceforge.net/mingw/MinGW-3.1.0-1.exe

  MinGW を用いて `mmul.c` をコンパイルする手順は以下の通りです.

  1. スタートメニューから〈アクセサリ〉→〈コマンドプロンプト〉を選択してコマンドプロンプトを開き, `mmul.c` があるディレクトリをカレントディレクトリにします.

  2. 次のように gcc コマンドを実行して `mmul.c` をコンパイルします:

     ```
     > gcc -IC:/silc-1.3/src/client -c mmul.c
     > gcc -LC:/silc-1.3/src/client -o mmul mmul.o -lsilc -lws2_32
     ```

     `-I` オプションと `-L` オプションにはそれぞれ `client.h` と `libsilc.a` があるディレクトリを指定します. また, WinSock2 ライブラリをリンクするために `-lws2_32` オプションが必要です.

     プログラムを繰返し修正してコンパイルする場合は `mmul.c` と同じディレクトリに次のような `Makefile` を作成しておくと便利です:

     ```
     all: mmul

     SILC=    C:/silc-1.3
     ```

```
CC=      gcc
CFLAGS=  -I$(SILC)/src/client
LDFLAGS= -L$(SILC)/src/client
LIBS=    -lws2_32

mmul: mmul.c
        $(CC) $(CFLAGS) -c mmul.c
        $(CC) $(LDFLAGS) -o $@ mmul.o -lsilc $(LIBS)
```

このファイルを用いて `mmul.c` をコンパイルするには次のように make コマンドを実行します:

```
> make
```

3. もうひとつ別のコマンドプロンプトを開いて SILC サーバを起動します:

```
> C:
> cd ¥silc-1.3¥src¥server
> server
```

次に, 最初のコマンドプロンプトでユーザプログラムを実行します:

```
> mmul
connected to kajiyama on port 1639
number of formats = 4
  0: "SILC:dense (column major)"
  1: "SILC:Band"
  2: "SILC:CRS"
  3: "SILC:JDS"
Request: >A
Response: 200 OK
Request: :9:X = A * A
Response: 200 OK
Request: <X
Response: 200 OK
A =
 1.000000e+000 3.000000e+000
 2.000000e+000 4.000000e+000
A * A =
 7.000000e+000 1.500000e+001
 1.000000e+001 2.200000e+001
>
```

上記のような実行結果が得られれば正常に動作しています.

著作権および利用条件

本ソフトウェアの著作権は SSI プロジェクトにあります. あなたは所定の利用条件にしたがって自由に本ソフトウェアを利用できます. 本ソフトウェアは無保証です. 本ソフトウェアの詳細な利用条件については `LICENSE` ファイルをご覧下さい.

また、本パッケージには以下の著作物が含まれています. これらの著作物は当該著作物の利用条件にしたがって利用して下さい.

- `src¥server¥libblas.dll`, `src¥server¥liblapack.dll`

  BLAS および LAPACK: The Netlib (http://www.netlib.org/) より.

- `src¥server¥dgefa.f`, `src¥server¥dgesl.f`

  LINPACK: The Netlib (http://www.netlib.org/linpack/) より.

- `src¥lib¥mt¥mt19937ar.c`, `src¥lib¥mt¥mt19937ar.h`

  疑似乱数生成器 Mersenne Twister: 原本は以下の場所から入手できます.

  http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html

連絡先

バグ報告, ご要望, ご意見等をお寄せ下さい. 下記の宛先までお気軽にどうぞ.

SSI プロジェクト <devel **at** ssisc.org>

$Id: README.win32.ja,v 1.14 2007/10/31 05:18:48 kajiyama Exp $

console プログラムにおける命令記述言語の拡張

最終更新日: 2007年10月31日

はじめに

この文書では，SILC のサンプルプログラムの1つである console プログラム (`src/client/console.c`) における SILC の数式 (命令記述言語) の拡張仕様について述べる．このプログラムは，(1) SILC の数式を対話的に実行すること，(2) たくさんの数式をファイルから読み込んで一括実行することができる．また，SILC の命令記述言語の仕様を拡張しており，条件分岐や反復などを記述できる．このプログラムを用いることにより，C や Fortran など他のプログラミング言語の処理系がなくても SILC のユーザプログラムを作成できる．

条件分岐

条件分岐は以下の形式で記述する:

```
if (cond_expr) {
  stmt; ...
} else if (cond_expr) {
  stmt; ...
} else {
  stmt; ...
}
```

`cond_expr` には後述の条件式を指定する． `stmt` は任意の文を表す．else 節は省略できる．中括弧は省略できない．

反復，continue 文，break 文

反復は以下の形式で記述する:

```
while (cond_expr) {
  stmt; ...
}
```

`cond_expr` には後述の条件式を指定する．中括弧は省略できない．また， continue 文と break 文が利用できる．

条件式

条件式 (`cond_expr`) は以下の比較演算子から成る.

- expr < expr
- expr > expr
- expr <= expr
- expr >= expr
- expr == expr
- expr != expr

`expr` には SILC の命令記述言語で記述された式を指定する. この式の値はスカラーでなければならない. スカラー以外 (ベクトル, 行列など) の値を取る式を与えると実行時エラーとなる. さらに, 上記の条件式は次の論理演算子および括弧により結合できる.

- cond_expr or cond_expr
- cond_expr and cond_expr
- not cond_expr
- (cond_expr)

比較演算子および論理演算子は値として真理値を取る. 真理値を取らない式は条件式として利用できない. また, 真理値を表す定数として `true` および `false` が利用できる.

console プログラムにおける比較演算子の処理方法は次の通りである.

1. 比較演算子の左辺および右辺に指定された式をそれぞれ一時変数 `_` に対する代入文として実行する:

   ```
   SILC_EXEC("_ = expr")
   ```

2. 一時変数の値を SILC サーバから受け取る:

   ```
   SILC_GET(&tmp, "_")
   ```

3. 受け取った値をクライアント側で比較する.

拡張システム文

SILC の命令記述言語で定義されているシステム文に加えて, 以下の拡張システム文が利用できる.

- load "ファイル名", 変数名

  Matrix Market 形式の行列データファイルから行列やベクトルなどのデータを読み込んでサーバに送る. 例:

  ```
  load "filename.mtx", A
  load "filename.mtx", b
  ```

- save "ファイル名", 変数名

  サーバからデータを受け取って Matrix Market 形式でファイルに保存する. 例:

  ```
  save "filename.mtx", x
  ```

- pprint expr

  サーバから式 `expr` の値を受け取ってクライアント側の標準出力に印字する.

- message "文字列"

  指定された文字列をクライアント側の標準出力に印字する.

その他

条件分岐, 反復, および拡張システム文はクライアント側で実行する. その他の文 (SILC の命令記述言語における文) はサーバ側で実行する. すなわち, データはすべて SILC サーバ側で管理する (console プログラムにはデータ管理機構はない).

文はセミコロンで区切ってつなげることができる. 行末のセミコロンは省略可.

行末に `¥` を置くと直後の改行記号を読み飛ばす. 例えば, 以下の文は1行とみなされる:

```
B = {1, 2, 3; ¥
     4, 5, 6; ¥
     7, 8, 9}
```

`#` から行末まではコメントとみなして読み飛ばす.

利用例

上述の拡張命令記述言語で実現した共役勾配 (Conjugate Gradient, CG) 法のプログラムを以下に示す:

```
# 三重対角行列 A とベクトル b を作る
n = 400
A = diag(2.0 * ones(n, 1)) - diag(ones(n-1, 1), 1) - diag(ones(n-1, 1), -1)
b = A * (-ones(n, 1))

# 連立一次方程式 Ax=b を CG 法で解く
rho_old = 1.0
p = zeros(n, 1)
x = zeros(n, 1)
r = b
bnrm2 = 1.0 / norm2(b)
iter = 1
while (iter <= n) {
  rho = r' * r
  beta = rho / rho_old
  p = r + beta * p
  q = A * p
  alpha = rho / (p' * q)
  r = r - alpha * q
  nrm2 = norm2(r) * bnrm2
  x = x + alpha * p
  if (nrm2 <= 1.0e-12) {
    break
  }
  rho_old = rho
  iter += 1
```

```
}

# 解 x をファイルに保存する
save "sol.mtx", x

# 反復回数を表示する
message "number of iterations:"
pprint iter
```

実行方法: 上記のプログラムをファイル (例えば `silc_cg.txt`) に保存し，SILC サーバを起動した上で次のように console プログラムを実行する:

```
console silc_cg.txt
```

反復回数 (number of iterations) が200回で終了すれば正常動作している.

改訂履歴

- 2007年10月31日
  - SILC v1.3 の新しいディレクトリ構成に合わせて文中のファイル名を修正.
- 2007年2月13日
  - 最初のバージョン.

$Id: README.console.ja,v 1.5 2007/10/31 05:18:48 kajiyama Exp $

SILC利用者マニュアル

SSI プロジェクト

2007年10月31日改訂（SILC v1.3 対応）

目次

1. はじめに

本文書では，SILC の応用プログラム（ユーザプログラム）を作成するために必要となる情報を提供する．まず SILC サーバのコンパイル方法と実行方法について述べる．次に C 言語と Fortran によるユーザプログラムの開発方法について述べる．さらに SILC の数式 (命令記述言語) について解説し，最後に API の詳細をまとめる．

2. SILC サーバ

2.1. SILC サーバのコンパイル

SILC サーバをコンパイルして動かすには以下のソフトウェアがインストールされた Unix/Linux 環境または Windows 環境が必要である．

- C/Fortran コンパイラ

- GNU Make
- GNU Bison
- GNU Flex

また, 以下の行列計算ライブラリを SILC サーバに組み込んで利用できる.

- BLAS, LAPACK (http://www.netlib.org/)
- Lis 1.0.2 (http://www.ssisc.org/lis/)
- FFTSS (http://www.ssisc.org/fftss/)

SILC の動作が確認されている計算環境（括弧内は後述する `make.inc` の作成例を示す `inc/` ディレクトリ内のファイル名）, OS, およびコンパイラは以下の通りである.「OpenMP」の欄は当該環境における OpenMP のサポートの有無を示す.

表 1. テスト環境

計算環境	OS	コンパイラ	OpenMP
Sun Fire 3800 (make.sunfire)	Solaris 9 (sparc)	Sun ONE Studio 7	Yes
SGI Altix 3700 (make.altix)	Red Hat Linux Advanced Server 2.1	Intel C 9.1 Intel Fortran 9.1	Yes
IBM eServer xSeries 335 (make.linux-icc-32)	Red Hat Linux 8.0	Intel C 9.0 Intel Fortran 9.0	Yes
Dell PowerEdge SC 1420 (make.linux-icc-64)	Fedora Core 4	Intel C 9.0 (EM64T) Intel Fortran 9.0 (EM64T)	Yes
IBM OpenPower 710 (make.openpower)	SuSE Linux Enterprise Server 9 (ppc)	IBM XL C 7.0 IBM XL Fortran 9.1	Yes
Apple PowerMac G5 (make.g5)	Mac OS X 10.4.2	IBM XL C 6.0 IBM XL Fortran 8.1	Yes
NEC SX-6i (make.sx)	SUPER-UX 13.1 SX-6	C++/SX 1.0 for SX-6 FORTRAN90/SX 2.0 for SX-6	No
Panasonic CF-R3 (make.gcc)	Fedora Core 3	GCC 3.4.2	No
IBM ThinkPad T42 (make.gcc)	KNOPPIX 4.0 LinuxTag Japanese Edition	GCC 3.3.6	No
Dell Dimension 84000 (make.mingw)	Microsoft Windows XP Professional SP2	MinGW (GCC 3.2.3)	No

SILC のソースコード（`silc-1.3.tar.gz`）は以下の場所から入手できる.

http://www.ssisc.org/silc/

`silc-1.3.tar.gz` からソースコードを取り出すには次のコマンドを実行する. これによりカレントディレクトリの `silc-1.3/src/` ディレクトリ内に SILC のソースコードが展開される.

```
$ gzip -cd silc-1.3.tar.gz | tar xvf -
```

SILC をコンパイルするには, 計算環境に合わせた内容の `make.inc` というファイルを `silc-1.3/src/` ディレクトリ内に作成する必要がある. 上述のテスト環境向けに作成した `make.inc` ファ

イルのサンプルが `silc-1.3/src/inc/` ディレクトリ内にある．サンプルファイル名と想定する計算環境，OS，コンパイラの対応関係は表 1 の通りである．

`make.inc` ファイルには，**make** コマンドの入力ファイルとなる `Makefile` で用いられるマクロ変数を記述する．マクロ変数には大きく分けて SILC に依存しない一般的なもの（表 2）と SILC 固有のマクロ変数（表 3）がある．

表 2. 一般的なマクロ変数

`CC`	C コンパイラのコマンド名
`FC`	Fortran コンパイラのコマンド名
`LINK.f`	Fortran プログラムのリンクに用いるリンカのコマンド名（デフォルトは `FC` に指定したコマンド名）
`BISON`	GNU Bison のコマンド名
`FLEX`	GNU Flex のコマンド名
`CFLAGS`	C のコンパイラオプション
`FFLAGS`	Fortran のコンパイラオプション
`LDFLAGS`	C/Fortran コンパイラのリンカオプション
`SHAREDCFLAGS`	共有ライブラリのコンパイルに必要な C コンパイラオプション
`SHAREDFFLAGS`	共有ライブラリのコンパイルに必要な Fortran コンパイラオプション
`SHAREDLDFLAGS`	共有ライブラリのリンクに必要な C/Fortran コンパイラのリンカオプション
`RANLIB`	**ranlib** コマンドの名前（**ranlib** コマンドを実行する必要のない環境では **echo** コマンド等を指定する）
`RM`	**rm** コマンドの名前（定義する必要があるのは一部の環境に限られる）

表 3. SILC 固有のマクロ変数

`OMPFLAGS`	OpenMP 並列化を有効にするための C/Fortran コンパイラオプション
`LIBS`	SILC サーバおよびサンプルプログラムをリンクするためのリンカオプション（リンクするライブラリ等を指定）
`PLATFORM_MODULES`	特定のライブラリに依存する格納形式モジュールおよび計算モジュールのうち，SILC に組み込むものを列挙する．以下のモジュールが指定できる（括弧内は依存するライブラリ名）．各モジュールの詳細については4.7節 を参照． • `$(FORMAT_DIR)/sparse_band.so` (BLAS, LAPACK) • `$(MODULE_DIR)/blasmodule.so` (BLAS, LAPACK) • `$(MODULE_DIR)/leq_lis.so` (Lis) • `$(MODULE_DIR)/fftss.so` (FFTSS) • `$(MODULE_DIR)/linpackmodule.so` (LINPACK)
	特定のライブラリに依存するサンプルプログラムのうちコンパイルするものを列挙する．以下のサンプルプログラムが指定できる（括弧内は依存するライブラリ名）． • `test_dense` (BLAS) • `test_dense_sa` (BLAS)

PLATFORM_TESTS	• test_dot • test_dot_sa (BLAS) • test_band • test_band_sa (BLAS, LAPACK) • mm_band (BLAS) • mm_band_sa (BLAS, LAPACK) • mm_lis (Lis)
BLAS_DRIVER	BLAS ドライバのファイル名（後述）
BLAS_CFLAGS	BLAS を呼ぶプログラムをコンパイルするためのCコンパイラオプション
BLAS_LIBS	BLAS を呼ぶプログラムをリンクするためのリンカオプション（リンクするライブラリ等を指定）
LAPACK_DRIVER	LAPACK ドライバのファイル名（後述）
LAPACK_CFLAGS	LAPACK を呼ぶプログラムをコンパイルするためのCコンパイラオプション
LAPACK_LIBS	LAPACK を呼ぶプログラムをリンクするためのリンカオプション（リンクするライブラリ等を指定）
LIS_CFLAGS	Lis を呼ぶプログラムをコンパイルするための C コンパイラオプション
LIS_LD	Lis を呼ぶプログラムをリンクするのに用いるリンカのコマンド名
LIS_LIBS	Lis を呼ぶプログラムをリンクするためのリンカオプション（リンクするライブラリ等を指定）
LINPACK_LIBS	LINPACK を呼ぶプログラムをリンクするためのリンカオプション（リンクするライブラリ等を指定）

BLAS および LAPACK は，SILC サーバとサンプルプログラムからドライバを介して呼び出される．ドライバには以下の3種がある．用いる BLAS/LAPACK のバージョンに応じて適当なドライバを1つ選んで利用する．

- Intel Math Kernel Library（MKL）用ドライバ

  `blas_intelmkl.c`, `lapack_intelmkl.c`

- Sun Performance Library 用ドライバ

  `blas_sunperf.c`, `lapack_sunperf.c`

- 汎用ドライバ

  `blas_dumb.c`, `lapack_dumb.c`

用いるドライバ，コンパイラオプション，およびリンカオプション（リンクするライブラリ等）を指定するには `make.inc` ファイルに以下のマクロ変数を定義する．

BLAS_DRIVER	BLAS ドライバのファイル名
BLAS_CFLAGS	BLAS 用の C コンパイラオプション
BLAS_LIBS	BLAS 用のリンカオプション
LAPACK_DRIVER	LAPACK ドライバのファイル名
LAPACK_CFLAGS	LAPACK 用の C コンパイラオプション

LAPACK_LIBS　　LAPACK 用のリンカオプション

汎用 BLAS/LAPACK ドライバを用いる場合のマクロ定義例を以下に示す.

```
BLAS_DRIVER=    blas_dumb.c
BLAS_CFLAGS=
BLAS_LIBS=      /opt/LAPACK/blas.a

LAPACK_DRIVER=  lapack_dumb.c
LAPACK_CFLAGS=  $(BLAS_CFLAGS)
LAPACK_LIBS=    /opt/LAPACK/lapack.a $(BLAS_LIBS)
```

silc-1.3/src/inc/ ディレクトリ内の make.inc の作成例は -DDEBUG オプションを付けてコンパイルする設定になっているため, SILC サーバとユーザプログラムの実行時に多量のメッセージが出力される. make.inc を修正してこのオプションを削除またはコメントアウトすればメッセージの出力を抑えることができる.

silc-1.3/src/ ディレクトリ内に make.inc を作成した後, 同じディレクトリで次のように **make** コマンドを実行して SILC サーバをコンパイルする. SILC のサンプルプログラムも一緒にコンパイルされる.

```
$ make
```

2.2. SILC サーバの起動と終了

ユーザプログラムを実行する前に SILC サーバを起動しておく必要がある. SILC サーバを起動するには次のように入力する(注: SILC サーバの実行可能形式ファイルがあるディレクトリをカレントディレクトリにする必要がある). SILC サーバはフォアグラウンドで動作する.

```
$ cd src/server
$ ./server
```

共有メモリ並列環境では OpenMP で並列化された SILC サーバが利用できる. スレッド数を制御するには環境変数 OMP_NUM_THREADS を設定する. 例:

```
$ env OMP_NUM_THREADS=4 ./server
```

特に指定しない限り SILC サーバが用いるポート番号は毎回変化する. ポート番号を指定するには環境変数 SILC_PORT を設定する. 例:

```
$ env SILC_PORT=32000 ./server
```

SILC サーバは起動後に自身のホスト名とポート番号を ~/.silc に書き出す. 後述するクライアントルーチン SILC_INIT(3.1節)は, このファイルに記述されたホスト名とポート番号に従ってサーバへの接続を確立する. ~/.silc の内容は以下の通りである.

```
ホスト名 ポート番号 [EOF]
```

SILC サーバとユーザプログラムをそれぞれファイルシステムの異なる計算環境で実行する場合はユーザプログラム側の ~/.silc を人手により作成する必要がある. 以下のように **echo** コマン

ドを利用するとよい:

```
$ echo ホスト名 ポート番号 > ~/.silc
```

SILC サーバを終了するには，**Ctrl-C** を入力するか，**kill** コマンドでサーバプロセスを終了させる．例:

```
$ kill -9 プロセス番号
```

3. ユーザプログラム

3.1. ユーザプログラムの構成

SILC のユーザプログラムは，ネットワークを介して SILC サーバに接続し，以下の関数を用いてサーバが管理するライブラリを利用する．これらの関数をクライアントルーチンと呼ぶ．引数の詳細は5節で述べる．

C 言語用クライアントルーチン:

- `SILC_INIT`

  SILC サーバに接続する．

- `SILC_PUT`

  行列やベクトルなどのデータをサーバに送る．

- `SILC_EXEC`

  サーバに計算を指示する．計算指示には SILC の命令記述言語（4節）で表された文字列（数式）を用いる．

- `SILC_GET`

  計算結果をサーバから受け取る．

- `SILC_FINALIZE`

  サーバとの接続を切る．

Fortran 用クライアントルーチン:

- `SILC_INIT`

  SILC サーバに接続する．

- `SILC_PUT_SCALAR`, `SILC_PUT_MATRIX`, `SILC_PUT_MATRIX_CRS` など

  行列やベクトルなどのデータをサーバに送る．C言語の場合とは異なり，送るデータの型に応じて異なる関数を用いる．

- SILC_EXEC

  サーバに計算を指示する. 計算指示には SILC の命令記述言語(4節)で表された文字列(数式)を用いる.

- SILC_GET_SCALAR, SILC_GET_MATRIX, SILC_GET_MATRIX_CRS など

  計算結果をサーバから受け取る. 上記の送信の場合と同様にデータ型別に異なる関数を用いる.

- SILC_FINALIZE

  サーバとの接続を切る.

これらのクライアントルーチンは C 言語用, Fortran 用ともに silc-1.3/src/client/libsilc.a に定義されている. このライブラリファイルをユーザプログラムにリンクすることにより上記のクライアントルーチン群が利用できる. libsilc.a は SILC サーバのコンパイル時に一緒に作成される.

また, クライアントルーチンの呼び出しに用いる構造体や定数(5節参照)は silc-1.3/src/client/client.h(C 言語用)および silc-1.3/src/client/fortran/client.h (Fortran 用)に定義されている.

なお, ユーザプログラムは逐次プログラムでもマルチスレッド化された並列プログラムでもよいが, クライアントルーチンの呼び出しは同一スレッドから行なわなければならない.

C 言語によるユーザプログラムの構成例(solve.c)を以下に示す. このプログラムは疎行列 A を係数とする連立一次方程式 Ax = b を解くプログラムの例である.

```
#include "client.h"

int main(int argc, char *argv[])
{
  silc_envelope_t object;  /* データの授受に用いる構造体 */
  double *value, *b, *x;
  int *index, *row;

  /* 疎行列 A (CRS 形式) およびベクトル b の生成 */

  SILC_INIT();

  object.v = value;
  object.type = SILC_MATRIX_TYPE;
  object.format = SILC_FORMAT_CRS;
  object.precision = SILC_DOUBLE;
  object.m = object.n = N;  /* 次数 */
  object.nnz = NNZ;  /* 非ゼロ要素数 */
  object.row = row;
  object.index = index;
  SILC_PUT("A", &object);

  object.v = b;
  object.type = SILC_COLUMN_VECTOR_TYPE;
  object.precision = SILC_DOUBLE;
  object.length = N;
  SILC_PUT("b", &object);

  /* 連立一次方程式 Ax = b の求解 */
  SILC_EXEC("x = A ¥¥ b");

  object.v = x;
  SILC_GET(&object, "x");
```

```
  SILC_FINALIZE();

  /* 解（ベクトル x）の出力 */
}
```

上記のユーザプログラムと同じ計算を行なう Fortran プログラムの構成例(solve.f)を以下に示す.

```
    INCLUDE 'client.h'

     REAL *8 VALUE(NNZ), B(N), X(N)
     INTEGER *4 ROW(N+1), INDEX(NNZ), IERR

C    疎行列 A（CRS 形式）およびベクトル b の生成

    CALL SILC_INIT(IERR)

     CALL SILC_PUT_MATRIX_CRS('A', VALUE, N, N, NNZ, ROW, INDEX,
    &                         SILC_DOUBLE, IERR)

     CALL SILC_PUT_COLUMN_VECTOR('b', B, N, SILC_DOUBLE, IERR)

C    連立一次方程式 Ax = b の求解
    CALL SILC_EXEC('x = A ¥ b', IERR)

     CALL SILC_GET_COLUMN_VECTOR('x', X, IERR)

     CALL SILC_FINALIZE(IERR)

C    解（ベクトル x）の出力
```

3.2. ユーザプログラムのコンパイル

ユーザプログラムをコンパイルするには，(1)ヘッダファイル(client.h)とライブラリファイル(libsilc.a)のあるディレクトリを指定するオプション，(2)リンクするライブラリ(libsilc.a および後述の追加ライブラリ)を指定するオプションをコンパイラに与える.

例えば，GNU C コンパイラを用いて前節の C プログラム(solve.c)をコンパイルするには次のように **gcc** コマンドを実行する．以下の例ではホームディレクトリ配下に ~/silc-1.3/src/ ディレクトリがあると仮定している.

```
$ gcc -I~/silc-1.3/src/client -c solve.c
$ gcc -L~/silc-1.3/src/client -o solve solve.o -lsilc
```

-I オプションと -L オプションはそれぞれ client.h と libsilc.a があるディレクトリを指定するオプション，-lsilc は libsilc.a のリンクを指示するオプションである.

なお，プログラムを繰返し修正してコンパイルする場合はユーザプログラムと同じディレクトリに次のような Makefile を作成すると便利である.

```
all: solve

CC=      gcc
CFLAGS=  -I$$HOME/silc-1.3/src/client
LDFLAGS= -L$$HOME/silc-1.3/src/client
LIBS=

solve: solve.c
        $(CC) $(CFLAGS) -c solve.c
```

```
	$(CC) $(LDFLAGS) -o $@ solve.o -lsilc $(LIBS)
```

このファイルを用いて `solve.c` をコンパイルするには次のように **make** コマンドを実行する.

```
$ make
```

また, GNU Fortran コンパイラを用いて前節の Fortran プログラム(`solve.f`)をコンパイルするには次のように **g77** コマンドを実行する.

```
$ g77 -I~/silc-1.3/src/client/fortran -fno-second-underscore -c solve.f
$ g77 -L~/silc-1.3/src/client -o solve solve.o -lsilc
```

C プログラムの場合と同様に次のような `Makefile` を作成しておくと便利である.

```
all: solve

FC=       g77 -fno-second-underscore
FFLAGS=   -I$$HOME/silc-1.3/src/client/fortran
LDFLAGS= -L$$HOME/silc-1.3/src/client
LIBS=

solve: solve.f
	$(FC) $(FFLAGS) -c solve.f
	$(FC) $(LDFLAGS) -o $@ solve.o -lsilc $(LIBS)
```

なお, いくつかの計算環境では `libsilc.a` と共にソケット通信機能を利用するためのライブラリをリンクする必要がある. 主な計算環境と追加ライブラリの有無を以下に示す. 上記の `Makefile` では `LIBS` に追加ライブラリを記述すればよい.

表 4. 主な計算環境と追加ライブラリの有無

計算環境	追加ライブラリ
Solaris	`-lsocket -lnsl`
GNU/Linux	なし
Microsoft Windows (MinGW)	`-lws2_32`
Mac OS X	なし

3.3. データ型と精度

ユーザプログラムと SILC サーバのあいだで授受できるデータ型には以下の5つがある. データ型の指定には括弧内の定数を用いる. これらの定数は `client.h` に定義されている(5節参照).

- スカラー(`SILC_SCALAR_TYPE`)
- 列ベクトル(`SILC_COLUMN_VECTOR_TYPE`)
- 行ベクトル(`SILC_ROW_VECTOR_TYPE`)
- 行列(`SILC_MATRIX_TYPE`)
- 3次元配列(`SILC_CUBIC_ARRAY_TYPE`)

利用可能なデータの精度を以下に示す．精度の指定には括弧内の定数を用いる．各精度に対応するC言語と Fortran のデータ型（精度）を示す．

表 5. 利用可能な精度

精度	C言語	Fortran
単精度整数（SILC_INT）	int	INTEGER*4
倍精度整数（SILC_LONG）	long	INTEGER*8
単精度実数（SILC_FLOAT）	float	REAL*4
倍精度実数（SILC_DOUBLE）	double	REAL*8
単精度複素数（SILC_COMPLEX）	float[a]	COMPLEX*8
倍精度複素数（SILC_DOUBLE_COMPLEX）	double[a]	COMPLEX*16
[a] C言語では，複素数の実部と虚部をそれぞれ実数で表す．長さ *N* の複素数配列は実部と虚部を交互に並べた長さ 2*N* の実数配列で表す．		

3.4. 行列の格納形式

行列を授受する場合はデータ型（SILC_MATRIX_TYPE）と共に格納形式を指定する必要がある．現在サポートしている行列の格納形式は以下の2種である．

- SILC_FORMAT_DENSE

  密行列（dense matrix）．Fortran の2次元配列に相当する格納形式．以下の属性値および配列から成る．

  m

  　　行数（整数型）

  n

  　　列数（整数型）

  value

  　　長さ m * n の配列（精度は任意）

  　　行列のすべての要素を Fortran の2次元配列の形式（列優先）で格納する（注：C言語の2次元配列は行優先）．

- SILC_FORMAT_CRS

  Compressed Row Storage（CRS）形式の疎行列（sparse matrix）．行列の要素のうち値が 0 でないもの（これを非零要素と呼ぶ）のみを保持する格納形式．以下の属性値および配列から成る．

  m

  　　行数（整数型）

  n

  　　列数（整数型）

nnz

非ゼロ要素数（整数型）

value

長さ *nnz* の配列（精度は任意）

非ゼロ要素のみを行優先ですき間なく格納する.

row

長さ $m + 1$ の単精度整数型配列

`row[0]` は1行目の最初の非ゼロ要素の配列 `value` における位置（添字），`row[1]` は2行目の最初の非ゼロ要素の添字というように，各行の非ゼロ要素の開始位置を格納する．`row[m]` の値は *nnz* とする.

index

長さ *nnz* の単精度整数型配列

配列 `value` における対応する添字の非ゼロ要素の列番号を保持する．例えば，行列のある行の5列目の要素が `value[7]` に格納されている場合，`index[7]` に列番号 5 を格納する.

例: 次の4行4列の行列を CRS 形式で表した場合の配列 value, row, および index の内容を以下に示す.

```
| 11  0  0  0 |
|  0 22  0  0 |
|  0 32 33  0 |
| 41  0  0 44 |
```

- value: 11, 22, 32, 33, 41, 44（長さ 6）
- row: 0, 1, 2, 4, 6（長さ 5）
- index: 1, 2, 2, 3, 1, 4（長さ 6）

4. 命令記述言語

クライアントルーチン `SILC_EXEC` の引数にはサーバに計算を指示するための文字列を与える．文字列の内容は SILC の命令記述言語で書かれた一種のプログラムである.

命令記述言語の実行単位は文（statement）であり，次の3種がある.

- 代入文

  変数への値の格納を指示する文．等号（=）の左辺には変数名（および添字），右辺には式（expression）を指定する．変数は代入により型宣言なしで作成でき，任意のデータ型の値を代入できる．既存の変数に新しい値を代入すると古い値は削除される.

- 手続き呼び出し文

  手続き(procedure)の呼び出しを指示する文.

- システム文

  SILC サーバを制御するための文. 現時点では prefer 文(4.3節)がある.

また, 複数の文をセミコロン(;)で結合して一括して `SILC_EXEC` に渡すことができる.

4.1. 代入文

代入文には次の2種がある.

- 単純代入文(simple assignment)

  "*変数名* `=` *式*" の形の代入文. 式の値を指定された名前の変数に代入する. 変数がなければ新たに定義し, 同名の変数があれば値を変更する. "`A = B`" のように式として変数名を与えた場合, 複製された値が代入される(変数 B の値のコピーが作られる).

- 累算代入文(augmented assignment)

  二項演算を伴う代入文であり, 以下の7種がある. 例えば, "*変数名* `+=` *式*" は "*変数名* `=` *変数名* `+` *式*" と等しい. 左辺の変数名は定義済みでなければならない.

  変数名 `+=` *式*

  : 加算

  変数名 `-=` *式*

  : 減算

  変数名 `*=` *式*

  : 乗算

  変数名 `/=` *式*

  : 除算

  変数名 `%=` *式*

  : 余り

  変数名 `*@=` *式*

  : 要素ごと乗算(elementwise multiplication)

  変数名 `/@=` *式*

  : 要素ごと除算(elementwise division)

代入文の右辺には式を指定する. 式の構成要素を以下に示す. ここで, x, y, e_1, e_2, ..., e_N は任意の式である. なお, 単項演算子および二項演算子のオペランドとして有効なデータ型については4.4節を, 利用可能な組込み関数については5.3節をそれぞれ参照.

- 単項演算子

  `x⁻`

  　　複素共役

  `x'`

  　　共役転置

  `x'⁻` または `x⁻'`

  　　転置

  `-x`

  　　符号反転（negation）

- 二項演算子

  `x + y`

  　　加算

  `x - y`

  　　減算

  `x * y`

  　　乗算

  `x / y`

  　　除算

  `x % y`

  　　余り

  `A ¥ b`

  　　連立一次方程式 $Ax = b$ の求解（A はN×N行列，b は列ベクトルまたはN×M行列）

  `x *@ y`

  　　要素ごと乗算（elementwise multiplication）

  `x /@ y`

  　　要素ごと除算（elementwise division）

- 関数呼び出し

  `f(e1, e2, ..., eN)`

  　　`f` は関数名，$e_1, e_2, ..., e_N$ は任意の式（引数）

- 連結（concatenation）

  {e_1, e_2, ..., e_N}

  列方向に連結する．e_1, e_2, ..., e_N がスカラーならば列ベクトルが得られる．

  {e_1; e_2; ...; e_N}

  行方向に連結する．e_1, e_2, ..., e_N がスカラーならば行ベクトルが得られる．

  {e_1;; e_2;; ...;; e_N}

  3次元配列を生成する．

- 範囲（range）

  {e_1:e_2}

  始点 e_1 から終点 e_2 までの整数を要素とする列ベクトルを生成する．e_1 と e_2 には値が整数となる式を指定する．

- リテラル

  小数点を伴う数値は倍精度実数（`SILC_DOUBLE`），それ以外の数値は単精度整数（`SILC_INT`）として扱われる．

- 定数

  `e`

  自然対数の底

  `i`

  虚数単位（複素数は 3 - 5 * i のように二項演算子を用いて得る）

  `pi`

  円周率

  `inf`

  無限大

- その他

  - 変数名は式である．
  - 括弧で演算子の結合順序を変更できる．
  - 式には添字（4.6節）を付加できる．

4.2. 手続き呼び出し文

手続き呼び出し文は，手続き名と引数の並びから成る．以下に例を示す．

`split(A, L, D, U)`

行列 `A` を下三角部分，対角部分，上三角部分に分け，それぞれ `L`, `D`, `U` に格納する．

手続き呼び出し文の引数には以下の3種がある．

in 引数

手続きへの入力となる引数．値は変更されない．

out 引数

手続きからの戻り値が格納される引数．

inout 引数

手続きへの入力となり，かつ戻り値が格納される引数．

上記の手続き split の例では，第1引数 は in 引数，残りは out 引数である．各引数の in/out/inout の区別，および引数として認められるデータ型は手続きごとに異なる．利用可能な組込み手続きについては5.3節を参照．

4.3. システム文

システム文は SILC サーバの動作を制御するための文であり，以下のものがある．

`prefer` *モジュール名*

指定したモジュールをモジュールリストの先頭に移動してモジュール関数の検索順序を変更する．モジュール関数の検索については4.7節を参照．

4.4. 演算子のオペランド

演算子のオペランドには任意の式を記述できるが，オペランドとして有効なデータ型は演算子ごとに異なる．演算子と有効なデータ型の組み合わせを表 6 と表 7 に示す．二項演算子のオペランドに関する制限は累算代入文（4.1節）にも適用される．

表 6. 単項演算子

複素共役[a]	スカラー，行ベクトル，列ベクトル，行列，3次元配列
共役転置[a][b]	スカラー，行ベクトル，列ベクトル，行列
転置[b]	スカラー，行ベクトル，列ベクトル，行列
符号反転	スカラー，行ベクトル，列ベクトル，行列，3次元配列
[a] オペランドの精度が整数または実数の場合，演算結果はオペランドと等しい． [b] オペランドがスカラーの場合，演算結果はオペランドと等しい．	

表 7. 二項演算子

演算子	左オペランド	右オペランド	演算結果
加算および減算	スカラー	スカラー	スカラー
	列ベクトル	列ベクトル	列ベクトル

	行ベクトル	行ベクトル	行ベクトル
	行列	行列	行列
	3次元配列	3次元配列	3次元配列
乗算	スカラー	スカラー	スカラー
	スカラー	列ベクトル	列ベクトル
	スカラー	行ベクトル	行ベクトル
	スカラー	行列	行列
	列ベクトル	スカラー	列ベクトル
	列ベクトル	行ベクトル	行列
	行ベクトル	スカラー	行ベクトル
	行ベクトル	列ベクトル	スカラー[a]
	行ベクトル	行列	行ベクトル
	行列	スカラー	行列
	行列	列ベクトル	列ベクトル
	行列	行列	行列
	3次元配列	スカラー	3次元配列
除算	スカラー	スカラー	スカラー
	行ベクトル	行列	列ベクトル
	行列	行列	行列
余り[b]	スカラー	スカラー	スカラー
連立一次方程式の求解	スカラー	スカラー	スカラー
	行列	列ベクトル	列ベクトル
	行列	行列	行列
要素ごと乗算および要素ごと除算	スカラー	スカラー	スカラー
	列ベクトル	列ベクトル	列ベクトル
	行ベクトル	行ベクトル	行ベクトル
	行列	行列	行列
	3次元配列	3次元配列	3次元配列

[a] 演算結果はベクトルの内積を表す.

[b] 2つのオペランドの精度は共に整数でなければならない.

4.5. 精度変換ルール

二項演算子のオペランドの精度が異なる場合, 計算前に精度を変換して2つのオペランドの精度を合わせる. 精度変換には以下のルールを適用する.

- 6つの精度のあいだに以下の順序関係を定める. ここで, x → y ならば y は x の上位精度であると言う. また, x → y かつ y → z ならば x → z である.

```
SILC_LONG   →  SILC_DOUBLE   →  SILC_DOUBLE_COMPLEX
    ↑              ↑                ↑
SILC_INT    →  SILC_FLOAT    →  SILC_COMPLEX
```

- 2つのオペランドの一方が他方の上位精度ならば，後者を前者の精度に変換する．

  例えば，一方のオペランドの精度が `SILC_DOUBLE`，他方が `SILC_FLOAT` ならば，前者は後者の上位精度なので後者を `SILC_DOUBLE` に変換する．

- どちらも他方の上位精度でなければ，双方を共通の上位精度に変換する．

  例えば，一方のオペランドの精度が `SILC_DOUBLE`，他方が `SILC_COMPLEX` ならば，双方を共通の上位精度 `SILC_DOUBLE_COMPLEX` に変換する．

4.6. 添字

代入文の左辺（変数名），および任意の式に添字を付加できる．添字は1から始まる．添字の記法を以下に示す．

- `A[`*x*`]`

  `A` はベクトル．

- `A[`*x*`,`*y*`]`

  `A` は行列．

- `A[`*x*`,`*y*`,`*z*`]`

  `A` は3次元配列．

ここで *x*, *y*, *z* は任意の式である．式の値はスカラー整数または整数の列ベクトルでなければならない（それ以外の値ならば実行時エラーとなる）．

添字の式として範囲（range）を用いる場合，始点と終点の式は各々省略できる．例えば，`a` を次数 *n* のベクトルとすれば，`a[:5]` と `a[1:5]`，`a[5:]` と `A[5:`*n*`]`，`a[:]` と `a[1:`*n*`]` はそれぞれ等しい．

添字を用いることのできる文脈とその機能，および用例を以下に示す．

表 8. 添字の文脈と機能

機能	文脈	用例
部分参照	代入文の右辺	`b = a[1:5]` ベクトル `a` の一部（長さ5）を新しいベクトルとして変数 `b` に代入する．
	関数の引数・手続きの in 引数	`y = f(A[1:5,1:5])` 行列 `A` の一部（5行5列）を関数 `f` への入力として渡す．`A` の値は不変．
制約代入	代入文の左辺	`A[1:5,1:5] = B` 行列 `A` の一部を行列 `B` の値で置き換える．
	手続きの out 引数	`p1(x, a[1:5])` ベクトル `a` の値を部分的に手続き `p1` で変更する．
部分参照＋制約代入	手続きの inout 引数	`p2(A[1:5,1:5])` 行列 `A` の一部を手続き `p2` への入力として渡して値を変更する．

4.7. モジュール

SILC 命令記述言語における演算子，関数，および手続きは，モジュール関数と呼ぶラッパーを介してライブラリ関数の呼び出しに置き換えられ実行される．モジュール関数は，いくつかのモジュールにまとめられている．用意されているモジュールとそれに含まれるモジュール関数の概要を以下に示す．

- coremodule

  行列以外のデータ型を処理するモジュール関数から成るモジュール．

- dense

  密行列を扱うモジュール関数から成るモジュール．

- sparse_crs

  CRS 形式の疎行列を扱うモジュール関数から成るモジュール．

- leq_lis

  反復解法ライブラリ Lis を呼び出すためのモジュール．

- leq_cg

  密行列および CRS 形式の疎行列に対する CG 法を実装するモジュール．

- leq_gs

  密行列に対する Gauss-Seidel 法を実装するモジュール．

- leq_lu

  密行列に対する LU 分解による線形解法を実装するモジュール．

- blasmodule

  BLAS（Basic Linear Algebra Subprograms）による行列ベクトル積などのモジュール関数を含むモジュール．

- fftss

  高速フーリエ変換ライブラリ FFTSS を呼び出すためのモジュール．

また，以下の実験的なモジュールがある．

- sparse_band

  LAPACK（Linear Algebra PACKage）の Band 形式および LU 分解を実装するモジュール．

- sparse_jds

  疎行列用の Jagged Diagonal Storage（JDS）形式を実装するモジュール．

- leq_smsamg

Super Matrix Solver AMG version 3 (株式会社ヴァイナス) を呼び出すためのモジュール.

- leq_mp

  多倍精度反復解法ライブラリ mp_crs を呼び出すためのモジュール. このモジュールを利用するには GNU MP ライブラリが必要である.

- linpackmodule

  Linpack ベンチマーク用のモジュール関数から成るモジュール.

SILC サーバは, 数式を構成する個々の演算子, 関数, および手続きについて上記のモジュールから適当なモジュール関数を1つ選択して実行する. いくつかの演算子, 関数, 手続きには, 対応するモジュール関数が複数存在する(例えば, leq_cg, leq_gs, leq_lu の3つにはそれぞれ連立一次方程式の求解演算子に対応するモジュール関数が含まれる). また, モジュール群は SILC サーバにおいてリスト構造で管理されている. SILC サーバは, リストの先頭から順にモジュール内のモジュール関数を調べてゆき, 最初に見つかったモジュール関数を演算に用いる. 同名の関数および手続きが複数のモジュールに定義されている場合の選択方法も同様であり, 最初に見つかったモジュール関数が用いられる.

リスト内のモジュールの順序を変更するには `SILC_EXEC` で prefer 文(4.3節)を用いる. prefer 文は指定されたモジュールをリストの先頭に移動する. これにより演算子, 関数, および手続きとモジュール関数の対応関係を容易に変更できる.

5. API リファレンス

5.1. C言語版クライアントルーチン

- ヘッダファイル `"client.h"`

  SILC のユーザプログラムを作成する際はこのヘッダファイルを用いる.

- `silc_envelope_t` 構造体

  `SILC_PUT` および `SILC_GET` の引数として用いる構造体. 授受するデータ型に応じて以下のメンバ変数を利用する.

  - 共通のメンバ変数:

    ```
    int type;
    ```

    データ型(3.3節). 以下のいずれかの値を設定する.

`SILC_SCALAR_TYPE`	(スカラー)
`SILC_ROW_VECTOR_TYPE`	(行ベクトル)
`SILC_COLUMN_VECTOR_TYPE`	(列ベクトル)
`SILC_MATRIX_TYPE`	(行列)
`SILC_CUBIC_ARRAY_TYPE`	(3次元配列)

    ```
    int precision;
    ```

    精度(3.3節). 以下のいずれかの値を設定する.

SILC_INT	（単精度整数）
SILC_LONG	（倍精度整数）
SILC_FLOAT	（単精度実数）
SILC_DOUBLE	（倍精度実数）
SILC_COMPLEX	（単精度複素数）
SILC_DOUBLE_COMPLEX	（倍精度複素数）

```
const char *format;
```

格納形式（3.4節）．行列（データ型が SILC_MATRIX_TYPE）の場合のみ，以下の値を設定する．

SILC_FORMAT_DENSE	（密行列）
SILC_FORMAT_CRS	（CRS 形式の疎行列）

o SILC_SCALAR_TYPEの場合：

```
void *v;
```

スカラー値へのポインタ（※）

o SILC_ROW_VECTOR_TYPE, SILC_COLUMN_VECTOR_TYPEの場合：

```
size_t length;
```

ベクトルの長さ

```
void *v;
```

ベクトル（1次元配列）へのポインタ（※）

o SILC_MATRIX_TYPE（SILC_FORMAT_DENSE）の場合：

```
size_t m, n;
```

次数

```
void *v;
```

行列（列優先の2次元配列）へのポインタ（※）

o SILC_MATRIX_TYPE（SILC_FORMAT_CRS）の場合：

```
size_t m, n;
```

次数

```
size_t nnz;
```

非零要素数

```
void *v;
```

非零要素の配列へのポインタ（※）

```
int *row;
```

行ポインタの配列へのポインタ(※)

```
int *index;
```

列インデックスの配列へのポインタ(※)

なお, `row` と `index` の要素は0オリジンで表す.

- `SILC_CUBIC_ARRAY_TYPE` の場合:

```
size_t l, m, n;
```

各次元の次数

```
void *v;
```

3次元配列へのポインタ(※)

`SILC_GET` の場合は※印のメンバ変数のみ設定する. `SILC_PUT` の場合はすべて設定する.

以下の関数はいずれも正常終了なら 0, エラーなら −1 を返す.

- `int SILC_INIT(void);`

  SILC サーバへの接続を確立する. ユーザプログラムの冒頭(サーバを利用し始める箇所)で呼び出す.

- `int SILC_FINALIZE(void);`

  SILC サーバへの接続を終了する. ユーザプログラムの最後(サーバの利用を終える箇所)で呼び出す.

- `int SILC_EXEC(const char *expr);`

  サーバに計算を指示する. expr には SILC の命令記述言語(4節)で記述したプログラムを文字列で与える.

- `int SILC_PUT(const char *name, silc_envelope_t *envelope);`

  データに名前をつけて SILC サーバに預ける. *name* にはデータの名前を文字列で与える. *envelope* には, 送信するデータに関する情報を設定した `silc_envelope_t` 構造体へのポインタを与える.

- `int SILC_GET(silc_envelope_t *envelope, const char *name);`

  名前を指定して SILC サーバからデータを受け取る. *envelope* には `silc_envelope_t` 構造体へのポインタを与える. *name* には受信するデータの名前を文字列で与える.

  受信データの格納領域を指定する方法には次の2つがある.

  a. ユーザが事前に必要なデータ領域を確保する

     `silc_envelope_t` 構造体のメンバ `v`(および CRS 形式の行列の場合はメンバ `row` と `index`)に受信データの格納領域へのポインタを設定して `SILC_GET` を呼び出すと, そ

の領域に受信データが格納される.

b. 必要なサイズの領域を SILC_GET に自動的に確保させる

silc_envelope_t 構造体のメンバ v に NULL を代入して SILC_GET を呼び出すと, 必要なサイズの領域が自動的に確保される. データが不要になったら, ユーザは v が指す領域(および CRS 形式の行列の場合はメンバ row と index が指す領域)を free 関数で開放しなければならない.

5.2. Fortran 版クライアントルーチン

- ヘッダファイル "client.h"

  SILC のユーザプログラムを作成する際はこのヘッダファイルを用いる.

後述のクライアントルーチンに共通する引数を以下に示す.

- INTEGER*4 precision

  精度(3.3節). 以下のいずれかの値を設定する.

SILC_INT	(単精度整数)
SILC_LONG	(倍精度整数)
SILC_FLOAT	(単精度実数)
SILC_DOUBLE	(倍精度実数)
SILC_COMPLEX	(単精度複素数)
SILC_DOUBLE_COMPLEX	(倍精度複素数)

- INTEGER*4 status

  サブルーチンの終了コード. 正常終了なら 0, エラーなら −1 を返す.

また, 下記の *type* には次のデータ型を用いる.

INTEGER*4	(SILC_INT の場合)
INTEGER*8	(SILC_LONG の場合)
REAL*4	(SILC_FLOAT の場合)
REAL*8	(SILC_DOUBLE の場合)
COMPLEX*8	(SILC_COMPLEX の場合)
COMPLEX*16	(SILC_DOUBLE_COMPLEX の場合)

- SILC_INIT(status)

  SILC サーバへの接続を確立する. ユーザプログラムの冒頭(サーバを利用し始める箇所)で呼び出す.

  出力:

  INTEGER*4 status

  終了コード

- SILC_FINALIZE(status)

SILC サーバへの接続を終了する. ユーザプログラムの最後(サーバの利用を終える箇所)で呼び出す.

出力:

`INTEGER*4 status`

終了コード

- `SILC_EXEC(expr, status)`

サーバに計算を指示する.

入力:

`CHARACTER*`*`size`* `expr`

命令記述言語(4節)による計算指示

出力:

`INTEGER*4 status`

終了コード

- `SILC_PUT_SCALAR(name, value, precision, status)`

スカラー値に名前を付けて SILC サーバに預ける.

入力:

`CHARACTER*`*`size`* `name`

名前

`type` `value`

スカラー値

`INTEGER*4 precision`

精度

出力:

`INTEGER*4 status`

終了コード

- `SILC_PUT_ROW_VECTOR(name, value, length, precision, status)`

行ベクトルに名前を付けて SILC サーバに預ける.

入力:

`CHARACTER*`*`size`* `name`

名前

type value(length)

ベクトルの要素の配列

INTEGER*4 length

ベクトルの長さ

INTEGER*4 precision

精度

出力:

INTEGER*4 status

終了コード

- SILC_PUT_COLUMN_VECTOR(name, value, length, precision, status)

  列ベクトルに名前を付けて SILC サーバに預ける.

  入力:

  CHARACTER**size* name

  名前

  type value(length)

  ベクトルの要素の配列

  INTEGER*4 length

  ベクトルの長さ

  INTEGER*4 precision

  精度

  出力:

  INTEGER*4 status

  終了コード

- SILC_PUT_MATRIX(name, value, m, n, precision, status)

  密行列に名前を付けて SILC サーバに預ける.

  入力:

  CHARACTER**size* name

  名前

type value(m, n)

行列の要素の配列

INTEGER*4 m, n

次数

INTEGER*4 precision

精度

出力:

INTEGER*4 status

終了コード

- SILC_PUT_MATRIX_CRS(name, value, m, n, nnz, row, index, precision, status)

疎行列(CRS 形式)に名前を付けて SILC サーバに預ける. *row* と *index* の要素は1オリジンでなければならない.

入力:

CHARACTER**size* name

名前

type value(nnz)

非零要素の配列

INTEGER*4 m, n

次数

INTEGER*4 nnz

非零要素数

INTEGER*4 row(m+1)

行ポインタの配列

INTEGER*4 index(nnz)

列インデックスの配列

INTEGER*4 precision

精度

出力:

INTEGER*4 status

終了コード

- `SILC_PUT_CUBIC_ARRAY(name, value, l, m, n, precision, status)`

  3次元配列に名前を付けて SILC サーバに預ける.

  入力:

  `CHARACTER*`*`size`* `name`

  名前

  `type` `value(l, m, n)`

  3次元配列

  `INTEGER*4 l, m, n`

  各次元の次数

  `INTEGER*4 precision`

  精度

  出力:

  `INTEGER*4 status`

  終了コード

- `SILC_GET_SCALAR(name, value, status)`

  指定した名前のスカラー値を SILC サーバから受け取る.

  入力:

  `CHARACTER*`*`size`* `name`

  名前

  出力:

  `type` `value`

  スカラー値

  `INTEGER*4 status`

  終了コード

- `SILC_GET_ROW_VECTOR(name, value, status)`

  指定した名前の行ベクトルを SILC サーバから受け取る. ユーザは, 配列 `value` を用意するために予めベクトルの長さ(*`length`*)を知っていなければならない.

  入力:

`CHARACTER*`*`size`* `name`

名前

出力:

`type` `value(`*`length`*`)`

ベクトルの要素の配列

`INTEGER*4 status`

終了コード

- `SILC_GET_COLUMN_VECTOR(name, value, status)`

  指定した名前の列ベクトルを SILC サーバから受け取る. ユーザは, 配列 `value` を用意するために予めベクトルの長さ(*`length`*)を知っていなければならない.

  入力:

  `CHARACTER*`*`size`* `name`

  名前

  出力:

  `type` `value(`*`length`*`)`

  ベクトルの要素の配列

  `INTEGER*4 status`

  終了コード

- `SILC_GET_MATRIX(name, value, status)`

  指定した名前の密行列を SILC サーバから受け取る. ユーザは, 配列 `value` を用意するために予め行列の次数(*`m`*, *`n`*)を知っていなければならない.

  入力:

  `CHARACTER*`*`size`* `name`

  名前

  出力:

  `type` `value(m, n)`

  行列の要素の配列

  `INTEGER*4 status`

  終了コード

- `SILC_GET_MATRIX_CRS(name, value, row, index, status)`

指定した名前の疎行列(CRS 形式)を SILC サーバから受け取る. ユーザは, 配列 `value`, `row`, および `index` を用意するために予め行列の次数(*m*, *n*)と非零要素数(*nnz*)を知っていなければならない. `row` と `index` の要素は1オリジンで表される.

入力:

`CHARACTER*`*`size`* `name`

名前

出力:

`type` `value(`*`nnz`*`)`

非零要素の配列

`INTEGER*4 row(`*`m`*`+1)`

列ポインタの配列

`INTEGER*4 index(`*`nnz`*`)`

行インデックスの配列

`INTEGER*4 status`

終了コード

- `SILC_GET_CUBIC_ARRAY(name, value, status)`

指定した名前の3次元配列を SILC サーバから受け取る. ユーザは, 配列 `value` を用意するために予め3次元配列の大きさ(*l*, *m*, *n*)を知っていなければならない.

入力:

`CHARACTER*`*`size`* `name`

名前

出力:

`type` `value(`*`l, m, n`*`)`

3次元配列

`INTEGER*4 status`

終了コード

5.3. 組込み関数・手続き

5.3.1. coremodule

- *スカラー* `dot(`*行ベクトル, 列ベクトル*`)` [関数]

行ベクトルと列ベクトルの内積を返す.

- *スカラー* `sqrt(`*スカラー*`)` [関数]
- *ベクトル* `sqrt(`*ベクトル*`)` [関数]

  与えられたスカラーまたはベクトルの各要素の平方根を返す.
- *スカラー* `norm2(`*ベクトル*`)` [関数]

  与えられたベクトルの2ノルムを返す.
- *スカラー* `length(`*ベクトル*`)` [関数]

  与えられたベクトルの長さ(要素数) を返す.
- *スカラー* `time()` [関数]

  現在の時刻を表す倍精度実数で返す.

5.3.2. dense モジュール

- *列ベクトル* `diagvec(`*行列*`)` [関数]

  与えられた行列の対角要素を列ベクトルにして返す.
- *行列* `zeros(`*スカラー*`)` [関数]
- *行列* `zeros(`*スカラー*`, `*スカラー*`)` [関数]

  全要素が 0(倍精度)の密行列を生成して返す. 引数が1つの場合は正方行列を生成する. 引数が2つの場合は第1引数が行数, 第2引数が列数を表す.
- *行列* `ones(`*スカラー*`)` [関数]
- *行列* `ones(`*スカラー*`, `*スカラー*`)` [関数]

  全要素が 1(倍精度)の密行列を生成して返す. 引数が1つの場合は正方行列を生成する. 引数が2つの場合は第1引数が行数, 第2引数が列数を表す.
- *行列* `rand(`*スカラー*`)` [関数]
- *行列* `rand(`*スカラー*`, `*スカラー*`)` [関数]

  倍精度実数の乱数から成る密行列を生成して返す. 引数が1つの場合は正方行列を生成する. 引数が2つの場合は第1引数が行数, 第2引数が列数を表す. 乱数列の初期化には後述の srand 手続きを用いる.
- *列ベクトル* `size(`*行列*`)` [関数]

  行列の次数(行数と列数)を長さ2の列ベクトルとして返す.
- *スカラー* `size(`*行列*`, `*スカラー*`)` [関数]

  行列の次数(行数または列数)を求める. 第2引数が 1 の場合は行数, 2 の場合は列数をそれぞれ返す.
- *行列* `full(`*スカラー*`, `*スカラー*`)` [関数]

倍精度の整数行列を生成して返す．第1引数は行数，第2引数は列数を表す．要素はすべて非零となるように選ばれる（ただし乱数ではない）．

- split(*行列* IN, *行列* OUT, *行列* OUT, *行列* OUT) [手続き]

  第1引数の行列を下三角部分，対角部分，上三角部分に分割する．各部分はそれぞれ第2〜4引数に指定した変数に格納される．

- srand(*スカラー* IN) [手続き]

  第1引数（整数）をシードにして乱数生成器（Mersenne Twister）を初期化する．

5.3.3. sparse_crs モジュール

- *行列* sparse(*ベクトル r*, *ベクトル c*, *ベクトル v*, *スカラー m*, *スカラー n*) [関数]

- *行列* sparse(*ベクトル r*, *ベクトル c*, *スカラー v*, *スカラー m*, *スカラー n*) [関数]

  CRS 形式の疎行列を生成して返す．第1引数のベクトル *r* は行番号のリスト，第2引数のベクトル *c* は列番号のリストを表す．第3引数 *v* がベクトルの場合，*r* と *c* の第 *k* 要素は *v* の第 *k* 要素を格納する行番号および列番号を表す．すなわち，*v* は行列の非零要素のリストを表す．3つのベクトル *r*, *c*, *v* の次数は同じでなければならない．*v* がスカラーの場合は行列のすべての非零要素が同じ値になる．このとき，*r* と *c* の次数は同じでなければならない．第4引数 *m* は生成する行列の行数，第5引数 *n* は列数を表す．行列の精度は *v* の精度と同じになる．

- *行列* zeros(*スカラー*) [関数]

- *行列* zeros(*スカラー*, *スカラー*) [関数]

  全要素が 0（倍精度）の CRS 形式の疎行列を生成して返す．引数が1つの場合は正方行列を生成する．引数が2つの場合は第1引数が行数，第2引数が列数を表す．

- *行列* eye(*スカラー*) [関数]

  指定された次数の単位行列（CRS 形式）を生成して返す．

- *行列* diag(*ベクトル*) [関数]

- *行列* diag(*ベクトル*, *スカラー*) [関数]

  対角行列を生成して返す．引数が1つの場合，与えられたベクトルの次数を *n* とすると，ベクトルの各要素を対角要素とする *n* 行 *n* 列の疎行列（CRS 形式）を生成する．引数が2つの場合，第2引数（以下 *k* とする）は対角からのオフセット値を表す．$k = 0$ ならば引数が1つの場合と等価であり，$k > 0$ ならば $(1, k+1)$ から右下にのびる上対角，$k < 0$ ならば $(1-k, 1)$ から右下にのびる下対角にベクトルの各要素が格納された行列を生成する．行列の次数は $n+k$ である．

- *列ベクトル* diagvec(*行列*) [関数]

  与えられた行列（CRS 形式）の対角要素を列ベクトルにして返す．

- *行列* fliplr(*行列*) [関数]

  与えられた行列（CRS 形式）の列を逆順に並べ替えて返す．

- *行列* flipud(*行列*) [関数]

与えられた行列（CRS 形式）の行を逆順に並べ替えて返す.

5.3.4. sparse_band モジュール

- *行列* `CRS(`*行列*`)` [関数]

  与えられた行列（band storage 形式）を CRS 形式に変換して返す.

- *行列* `band(`*行列*`)` [関数]

  与えられた行列（CRS 形式）を band storage 形式に変換して返す.

- *スカラー* `rcond(`*行列*`)` [関数]

  与えられた行列（band storage 形式）の条件数の逆数を返す.

A. 改訂履歴

- 2007年10月31日（SILC v1.3）
  - SILC サーバとユーザプログラムのコンパイル方法についての記述を追加した.
  - API リファレンスに新しいモジュール関数を追加した.
- 2006年11月12日（SILC v1.2）
  - 本文書のソースファイルを DocBook XML 形式に変更した.
  - いくつかのモジュール関数の記述を追加した.
- 2005年11月25日（SILC v1.1）
  - 若干の修正を加えた.
- 2005年9月20日（SILC v1.0）
  - SILC v1.0に合わせて作成.

```
$Id: users_ja.xml,v 1.12 2007/10/31 06:31:35 kajiyama Exp $
```