「ライブラリ利用：高性能プログラミング初級入門」Part Ⅱ

2013年3月12日（火）

東京大学 情報基盤センター
伊藤祥司

当講習会の目標

1. 60分後には，線形方程式（$Ax=b$）に対して，100種類近くのアルゴリズムで求解できるようになる．
2. 90分後には，線形方程式に対して，並列計算による求解ができるようになる．
3. 180分後には，Oakleaf-FX(FX)上で2種類の線形方程式求解ライブラリを使えるようになっている上に，反復解法に基づく求解アルゴリズムの特徴と注意点を理解している．

こんな夢のようなことを実現するのが当講習会！
その鍵となるのが・・・　LisとPETScだ！

当講習会の方針

1. 東京大学情報基盤センターのOakleaf-FXにおいて，疎行列向き線形方程式求解ライブラリを使用できるようにする（マシンの利用方法，システム環境の違いや，ライブラリのバージョンの違いなどのため，他のマシンでも同様の手順とは限らないことに注意）．
2. Lisの典型的な使用方法・習得要領について説明するが，ライブラリ記述の文法面等は取り上げない（基本的に詳細についてはマニュアルを参照）．
3. PETScの典型的な習得要領について説明するが，ライブラリ記述の文法面等は取り上げない（基本的に詳細についてはマニュアルを参照）．

当講習会で扱うLisとPETScの比較

赤文字：ライブラリ選択の観点

実習		Lis-1.3.0	PETSc-3.3-p4
○	線形方程式用反復解法	22種類	10種類程度* *User Manualの記述
○	前処理演算	10種類	10種類程度*
○	演算	実数演算	実数演算，複素数演算
○	実行形態	逐次演算，MPI，OpenMP，Hybrid（MPI+OpenMP）	（逐次演算，）MPI
	ユーザプログラム言語	C言語，Fortran	C言語，C++，Fortran
	その他の特長	4倍精度演算	様々なProfiling機能，簡単なGraphics機能
	線形方程式反復解法以外の解法，分野		
		固有値解法	線形方程式用直接解法，非線形解法，ODE

PETScライクな固有値解法ライブラリとしては，SLEPc, the Scalable Library for Eigenvalue Problem Computations [http://www.grycap.upv.es/slepc/] があるが，PETScの開発グループとは異なる．

講習内容

前半の内容

0. 準備，FXでの実習環境設定
1. Lisの概要
2. Lisインストールの準備作業
3. Lis逐次版のインストール
4. ジョブ実行（逐次版）
5. Lis使用に関する解説
6. 入力データ形式について
7. Lis並列版のインストール・実行
8. Lisテストプログラムの概要

後半の内容

9. PETScの概要
10. FXでのPETSc利用の準備
11. FXでのPETSc-逐次版利用例
12. FXでのPETSc-MPI版利用例
13. PETSc-複素数演算例題
14. PETSc使用に関する解説
15. PETScテストプログラムの概要
16. KSPに関する盲点
17. 両ライブラリの習得要領

0. 準備：Lisとテスト問題他の入手先

◆ Lis [http://www.ssisc.org/ lis/]

→ Downloadの「Version 1.3.xx」からソースコード，ユーザマニュアル をダウンロード

※ 当資料，当講習会では，lis-1.3.0 版を前提として説明，実習します．
[http://www.ssisc.org/lis/dl/lis-1.3.0.tar.gz]

◆ PETSc [http://http://www.mcs.anl.gov/petsc/]

→ Downloadのページからソースコード，ユーザマニュアルをダウンロード

※ 当資料，当講習会では，Oakleaf-FXにインストール済みの petsc-3.3-p4 版を前提として説明，実習します（3.0.0版等とは非互換があるので注意）．
[http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.3-p4.tar.gz]

◆ テスト問題 Matrix Market [http://math.nist.gov/MatrixMarket]

→ BrowseやSearchの項目のリンクをクリック

◆ SESNA : Survey and Evaluation System for Numerical Algorithms [http://sesna.jp]

求解アルゴリズムの体系的性能評価システム
→ Matrix Market の一部問題を Lisの一部を用いて
求解した結果情報のDBと求解結果の評価システム

0. 準備：Oakleaf-FX (FX)のマシン環境

◆ FXのシステム環境と当資料中の記述との対応：

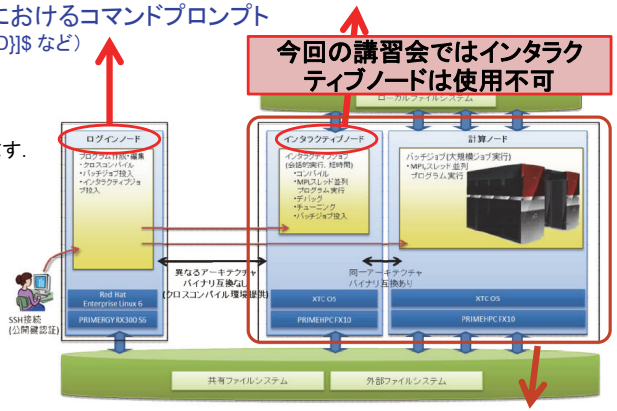
inter$ ：インタラクティブノードにおけるコマンドプロンプト
（実際は，[${USER}@a10tio297 ${PWD}]$ など）

$（または login$）：ログインノードにおけるコマンドプロンプト
（実際は，[${USER}@oakleaf-fx-2 ${PWD}]$ など）

ログインノードのコマンドプロンプトは，
基本的に $ と記述していますが，
インタラクティブジョブ実行する際など，
特に区別するときは login$ と記述しています．

◆ ジョブ実行用ノード使用時の注意点：

- インタラクティブノードへのログイン
  カレントディレクトリは継承される．
- 計算ノードへのバッチジョブ投入
  ジョブスクリプトのみが計算ノードへログインするイメージ．Oakleaf-FXのジョブ管理システムではカレントディレクトリへログインする（便利な仕様）．
- ジョブ実行用ノード共通
  ログインノードにて設定した環境（alias等）は，ジョブ実行用ノードには継承されない（あらためて設定する）．



「FX10 スーパーコンピュータシステム Oakleaf-FX 利用手引書 (ver. 2.4)」から

当資料中では「ジョブ実行用ノード」と総称

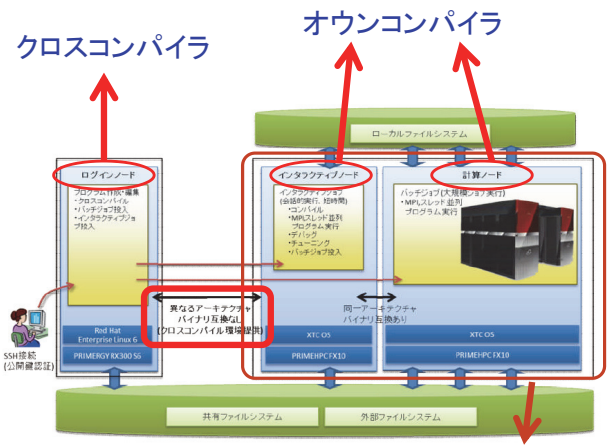
0. 準備：Oakleaf-FX (FX)のコンパイラ

◆ FXのコンパイラについて（当資料に関係するもののみ記載）：

	言語処理系	クロスコンパイラ	オウンコンパイラ
非MPIコード	Fortran 77/90	frtpx	frt
	C言語	fccpx	fcc
	C++	FCCpx	FCC
MPIコード	Fortran 77/90	mpifrtpx	mpifrt
	C言語	mpifccpx	mpifcc
	C++	mpiFCCpx	mpiFCC

クロスコンパイラ
ログインノードにてインタラクティブノード・計算ノード用の実行モジュールを作成する【通常はこちらを用いれば良い】
当資料の中では，Lisのインストール（サンプルプログラム作成含む），および，PETScのサンプルプログラム作成で用いている．

オウンコンパイラ
ログインノードの負荷が高いときや，コンパイルに相当な時間がかかるときなどに有効（コンパイラ自体も実行モジュールである）



「FX10 スーパーコンピュータシステム Oakleaf-FX 利用手引書 (ver. 2.4)」から

当資料中では「ジョブ実行用ノード」と総称

0. FXでの実習環境設定(1/5)

各自の$HOME下に実習用sampleファイル一式をコピー:

```
$ cd   [Enter]キー                          # ホームディレクトリ($HOME)へ移動
$ tar xf /home/t00003/sample.tar            # 実習用sampleファイル一式を入手・展開
$ ls -FR sample
sample:
library/  mat/  script/  setup.env  sh/

sample/library:                             # 今回の実習で使用するライブラリ等
lis-1.3.0.tar.gz  manual/  petsc-3.3-p4_samp/

sample/library/manual:                      # 今回の実習で使用するライブラリに対応した版のマニュアル
lis-manual-1.3.0_ja.pdf  petsc-manual-3.3-p4.pdf

sample/library/petsc-3.3-p4_samp:           # 今回の実習で使用するPETScの例題プログラム等
ex1.c  ex11.c@  ex11_mod.c  ex11_org.c  ex1_mod.c  ex23.c  makefile

sample/mat:                                 # テスト用入力データ(係数行列, 右辺項データ)
494_bus.mtx  add20_rhs1.mtx  add32_rhs1.mtx  memplus.mtx       nos3.rsa
494_bus.rsa  add32.mtx       add32_rhs2.mtx  memplus.rua
add20.mtx    add32.rua       fs_183_1.mtx    memplus_rhs1.mtx
add20.rua    add32_rhs1.dat  fs_183_1.rua    nos3.mtx
sample/script:                              # pjsub用スクリプト
pjsub_hyb.sh*  pjsub_mpi.sh*  pjsub_omp.sh*  pjsub_seq.sh*

sample/sh:                                  # 講習会向けJOB実行用ツール
isub*
```

以後 [Enter]キー
押下の記述は省略

- プロンプトと入力コマンドの部分は太字で表示してます.
- 画面表示は, 多少, 文字サイズを小さくしてます(当テキスト編集の都合上).
- 特に注意の無い限り, (# 記号から右部分の)青文字はコメントです.

0. FXでの実習環境設定(2/5)

各自の実習用環境をセットアップ:

```
$ ls sample
library  mat  script  setup.env  sh

$ source sample/setup.env                   # 実習環境のセットアップ
```

ログインの度に
実行する(☆1)

```
$ cat sample/setup.env                      # このような環境を設定
alias ls='ls -F'                            # .bashrc にて指定することも可能
alias lrt='ls -lrt'
alias cp='cp -i'
alias mv='mv -i'
alias rm='rm -i'
PATH=$PATH:$HOME/sample/sh
```


0. FXでの実習環境設定(3/5)

当講習会でのジョブ実行方法について(当講習会固有のコマンド):

※ isubツールは使用者側の責任で使用して下さい.

```
$ type isub          # これは ☆1のsetup.env を実行した上での話
isub is /home/t00003/sample/sh/isub
```

#ここで「 -bash: type: isub: not found 」などと表示されたら,
「0.FXでの実習環境設定(2/5)」の☆1を実行

```
$ isub               #擬似的なインタラクティブ実行にてバッチジョブを投入するコマンド
isub : Interact mode-like SUBmitting tool
  Usage:
     $ isub {-seq|-omp|-mpi|-hyb} [-th n] "command-line" [pjsub-option]
        option:
            -seq :  1 node   1 thread  job
            -omp :  1 node   8 threads job (default)
            -mpi :  4 nodes  1 thread  job (default)
            -hyb :  4 nodes  8 threads job (default)

            -th  :  Change the number of threads
                        n : The number of threads (default 8)

        output file (default):
            OUT  :  STDOUT + STDERR

                        Created by S.Itoh 20121218
```

両側をダブルクォーテーションで囲み, コマンド列を記述

isubコマンドを実行したディレクトリ上にOUTというファイルができる.
すでに同名のファイルが存在している場合には上書きされるので注意.
ただし, pjsub-optionで回避可能.

※ -seq オプションは, あくまでも"1ノード, 1スレッドジョブ用" という意味です
(非MPIジョブ用という意味では無い).

0. FXでの実習環境設定(4/5)

当講習会でのジョブ実行方法について(一般的なバッチジョブ方式):

```
$ ls sample/script
pjsub_hyb.sh*  pjsub_mpi.sh*  pjsub_omp.sh*  pjsub_seq.sh*

$ more sample/script/pjsub_seq.sh
#!/bin/sh
#PJM -L "rscgrp=tutorial"
#PJM -L "node=1"
#PJM -L "elapse=10:00"
#PJM -o OUT
#PJM -j

###
### For Lis examples
###

make check

# ./test1 testmat.mtx 0  sol1.txt  rsd1.txt
# ./test1 testmat.mtx 0  sol2.txt  rsd2.txt -i 3 -p 1
   ・・・ 以下省略 ・・・
```

このスクリプトにてpjsub実行したディレクトリ上にOUTというファイルができる.
すでに同名のファイルが存在している場合には上書きされるので注意.

上記の全スクリプトにてデフォルトは make checkである.
実習内容に応じ, スクリプト内のコメント記号を外したり, 各自でコマンドを書き加える.

#上記のスクリプトを実行モジュールがあるディレクトリにコピーし, 必要に応じスクリプトを書き換えた上で:

```
$ pjsub pjsub_seq.sh
```

いま, この時点では, これは実行しない

とすれば, 逐次(seq)実行のバッチジョブが投入される.

0. FXでの実習環境設定(5/5)

当講習会でのジョブ実行用ツール・スクリプトの設定内容：

・講習会中のキュー名：tutorial（実習用sampleは変更済み）
・講習会後のキュー名：lecture（ただし，当資料作成ではこちらのキューを使用）

```
$ more $HOME/sample/sh/isub
#!/bin/sh

MNODES=4
MTHREADS=8
#QUEUE="lecture"
QUEUE="tutorial"
OUTPUTFILE="OUT"

   ・・・ 以下省略 ・・・
```

- isubコマンドのデフォルト設定：
  ノード数　= 4 （mpi，hybの場合）
  スレッド数 = 8 （ompi，hybの場合）

```
$ more $HOME/sample/script/pjsub_seq.sh
#!/bin/sh                  # スクリプト内のシェルを指定
#PJM -L "rscgrp=tutorial"
#PJM -L "node=1"
#PJM -L "elapse=10:00"
#PJM -o OUT
#PJM -j

   ・・・ 以下省略 ・・・
```

- スクリプトの拡張子は任意
- ジョブ実行スクリプトの左端の“ #PJM ”記号は，これに続くハイフンで始まる文字列がpjsubコマンドのオプションであることを宣言．
- 赤太文字の項目をチェック（pjsubオプション「-L “node=1”」でノード数を指定）
- mpi，hybでは，ノード数=4 と設定
- omp，hybでは，スレッド数=8 と設定

1. Lisの概要（線形方程式版）

[http://www.ssisc.org/lis]

Lis (a Library of Iterative Solvers for linear systems)

- 開発元：JST CREST SSIプロジェクト（小武守，藤井，西田，長谷川 ）
  新バージョンの予定は無さそうだが，随時，マイナー修正が行われている．
  様々な計算機へのインストールが容易．
- 線形方程式求解アルゴリズムの
  逐次版，並列版（OpenMP：マルチスレッド環境，MPI：マルチプロセス環境）
  22 解法 × 11 前処理 × 3 スケーリング（“前処理無し”，“スケーリング無し” も含める）
- 11種類のデータ形式をサポート：　CSR(CRS), CSC(CCS), MSR, DIA, …
- 倍精度，4倍精度演算サポート

Component of Lis: [Lis-1.3.0 ver.]

(† Lis の特徴：国内研究者の成果，または，力を入れて研究している)

反復解法
非定常反復解法
CG, BiCG, CGS, BiCGStab, BiCGStab(*ℓ*), GPBiCG†, TFQMR, Orthomin(*m*), GMRES(*m*) BiCGSafe†, CR, BiCR†, CRS†, BiCRStab†, GPBiCR†, BiCRSafe†, FGMRES(*m*), IDR(*s*), MINRES
定常反復解法
Jacobi, Gauss-Seidel, SOR

前処理
None, (Point) Jacobi, ILU(*k*), SSOR, Hybrid†, I+S†, SAINV, SA-AMG†, Crout ILU, ILUT, Additive Schwarz
スケーリング（広義の前処理の一種）
None, $D^{-1}Ax = D^{-1}b.$ $D^{-1/2}AD^{-1/2}D^{1/2}\ x = D^{-1/2}b$

1. Lisの制限事項（1.3.0版）

前処理

- Jacobi, SSOR 以外の前処理が選択され，かつ行列A がCSR 形式でない場合，前処理作成時にCSR 形式の行列A が作成される．
- 線型方程式解法としてBiCG 法が選択された場合，SA-AMG 前処理は使用できない．
- SA-AMG 前処理はマルチスレッド環境では使用できない．
- SAINV 前処理の前処理行列作成部分は逐次実行される．

4倍精度演算

- 線型方程式解法のうち，Jacobi, Gauss-Seidel, SOR, IDR(s) 法では使用できない．
- 固有値解法のうち，CG, CR 法では使用できない．
- Hybrid 前処理での内部反復解法のうち，Jacobi, Gauss-Seidel, SOR 法では使用できない．
- I+S, SA-AMG 前処理では使用できない．

行列格納形式

- VBR 形式はマルチプロセス環境では使用できない．
- マルチプロセス環境において必要な配列を直接定義する場合は，CSR 形式で作成しなければならない．目的の格納形式を使用するには，lis_matrix_convert を使用してCSR 形式から変換する．

2. Lisインストールの準備

Lis実習環境の準備と逐次（seq）版の準備：

```
$ tar  zxf  $HOME/sample/library/lis-1.3.0.tar.gz
   # その他の方法の一例として：
  ( $ gunzip -c $HOME/sample/library/lis-1.3.0.tar.gz | tar xf - )

$ ls
lis-1.3.0/  sample/

$ cp -rp lis-1.3.0 lis-1.3.0_seq                    # Lis逐次（seq）版の環境準備

$ cd lis-1.3.0_seq
$ ls
AUTHORS     INSTALL      NEWS        build.sh*   configure.in  src/
COPYING     Makefile.am  README      config/     doc/          test/
ChangeLog   Makefile.in  aclocal.m4  configure*  include/      win/
```

- 以後，コンパイラ等の処理系のバージョンアップに起因して，出力情報が異なる場合もありますが，それらは適宜読み替えて下さい．
- ただし，コンパイラ等のバージョンが異なることによる非互換などはあり得ます．

3. Lis逐次版のインストール(1/5)

Lis逐次(seq)版のconfigure:

```
$ pwd
/home/t00003/lis-1.3.0_seq
$ ./configure --enable-saamg TARGET=fujitsu_fx10_cross # SAAMG前処理を有効にする
  ・・・ 途中省略 ・・・
  ・・・ 出力の最後部 ・・・

 Enable Fortran Interface       = no
 Enable SA-AMG Preconditioner   = yes
 Enable MPI                     = no
 Enable OpenMP                  =
 Enable Quad Precision          = no
 Enable Long Long Int           = no
 Enable Dynamic Linking         = no
 Enable Profiling               = no

 C Compiler         = fccpx
 C Flags            = -Kfast,ocl,preex -w -DHAVE_CONFIG_H
 C Libraries        = -lm
 F90 Compiler       = frtpx
 F90 Flags          = -Kfast,ocl,preex -Cpp -fs -DZERO_ORIGIN=1
 F90 Libraries      =
```

Lis開発側で実績のあるマシンの場合は，TARGETにて計算機環境を指定し，最適化可能だが，実績の無いマシンでも，大抵はインストール＆使用可能である．

赤文字の5項目をチェック

● configure実行後は，作業ディレクトリやパスを変更しないように(生成されるMakefile中では絶対パスが用いられている)．

3. Lis逐次版のインストール(2/5)

Lis逐次(seq)版のmake:

```
$ make
  ・・・ 途中省略 ・・・
  ・・・ 出力の最後部 ・・・

/bin/sh ../libtool --mode=link frtpx   -mlcmain=main -o etest5 -lm   etest5.o -L../src -
llis -lm
libtool: link: frtpx -mlcmain=main -o etest5 etest5.o  -L/home/t00003/lis-1.3.0_seq/src
/home/t00003/lis-1.3.0_seq/src/.libs/liblis.a -lm
fccpx  -DHAVE_CONFIG_H -I. -I../include  -I.. -I../include   -Kfast,ocl,preex -w -c
etest6.c
/bin/sh ../libtool --mode=link frtpx   -mlcmain=main -o etest6 -lm   etest6.o -L../src -
llis -lm
libtool: link: frtpx -mlcmain=main -o etest6 etest6.o  -L/home/t00003/lis-1.3.0_seq/src
/home/t00003/lis-1.3.0_seq/src/.libs/liblis.a -lm
make[1]: ディレクトリ `/home/t00003/lis-1.3.0_seq/test' から出ます
make[1]: ディレクトリ `/home/t00003/lis-1.3.0_seq' に入ります
make[1]: `all-am' に対して行うべき事はありません．
make[1]: ディレクトリ `/home/t00003/lis-1.3.0_seq' から出ます
```

最後部でErrorが出ていなければOK

3. Lis逐次版のインストール(3/5)

Lis逐次(seq)版のインストールチェック1(isubコマンドによるmake check):

```
$ isub  -seq  "make check"      #クロスコンパイルのため，ログインノードでのジョブ実行は不可能
make check                      #実行させたいコマンド列の両側をダブルクォーテーションで囲む
[INFO] PJM 0000 pjsub Job 437631 submitted.
                          ジョブID
$ pjstat                                        # ジョブ実行中の表示例
Oakleaf-FX scheduled stop time: 2012/12/28(Fri) 09:00:00 (Remain: 14days 17:03:11)

JOB_ID  JOB_NAME   STATUS PROJECT RSCGROUP START_DATE      ELAPSE      TOKEN NODE:COORD
437631  isub       QUEUED gt00    lecture  --/-- --:--:--  00:00:00    (0.2) 1

$ pjstat                                        # ジョブ終了後の表示例
Oakleaf-FX scheduled stop time: 2012/12/28(Fri) 09:00:00 (Remain: 14days 17:03:06)

No unfinished job found.

$ type lrt
lrt is aliased to `ls -lrt'

$ lrt
 ・・・ 省略 ・・・
-rwxr-xr-x  1 t00003 gt00  76621 12月 13 15:41 2012 config.status*
-rw-r--r--  1 t00003 gt00  31775 12月 13 15:41 2012 config.log
-rw-r--r--  1 t00003 gt00  48946 12月 13 15:41 2012 Makefile
drwxr-xr-x 12 t00003 gt00   4096 12月 13 15:45 2012 src/
drwxr-xr-x  4 t00003 gt00   4096 12月 13 15:56 2012 test/
-rw-r--r--  1 t00003 gt00   4684 12月 13 15:56 2012 OUT      # make check の実行結果
```


3. Lis逐次版のインストール(4/5)

Lis逐次(seq)版のインストールチェック2(isubコマンドによるmake check):

```
$ cat OUT
  ・・・ 出力の途中部分 ・・・

make  check-TESTS
make[2]: Entering directory `/home/t00003/lis-1.3.0_seq/test'
=== Running test test.sh

checking linear solvers...

number of processes = 1
matrix size = 100 x 100 (460 nonzero entries)
initial vector x = 0
precision : double
solver    : BiCG 2
precon    : none
storage   : CSR
lis_solve : normal end

BiCG: number of iterations     = 15 (double = 15, quad = 0)
BiCG: elapsed time             = 4.179478e-04 sec.
BiCG:   preconditioner         = 5.984306e-05 sec.
BiCG:     matrix creation      = 3.099442e-05 sec.
BiCG:   linear solver          = 3.581047e-04 sec.
BiCG: relative residual 2-norm = 1.689581e-16

checking eigensolvers...
 ・・・
```

赤文字の項目をチェック

下線の項目は計測時間なので，一致して無くても良い

3. Lis逐次版のインストール(5/5)

Lis逐次(seq)版のインストールチェック(番外編:pjsubコマンドによるmake check):

$HOME/sample/script/pjsub_seq.sh をコピーして
下記のとおり実行すれば良いが，ここでは実行しなくとも良い．

```
$ pjsub pjsub_seq.sh          #クロスコンパイルのため，ログインノードでのジョブ実行は不可能
[INFO] PJM 0000 pjsub Job 437667 submitted.

$ lrt
 ･･･ 省略 ･･･
-rwxr-xr-x  1 t00003 gt00  76621 12月 13 15:41 2012 config.status*
-rw-r--r--  1 t00003 gt00  31775 12月 13 15:41 2012 config.log
-rw-r--r--  1 t00003 gt00  48946 12月 13 15:41 2012 Makefile
drwxr-xr-x 12 t00003 gt00   4096 12月 13 15:45 2012 src/
-rwxr-xr-x  1 t00003 gt00    628 12月 13 16:21 2012 pjsub_seq.sh*
drwxr-xr-x  4 t00003 gt00   4096 12月 13 16:21 2012 test/
-rw-r--r--  1 t00003 gt00   4684 12月 13 16:21 2012 OUT      # make check の実行結果

$ cat OUT
  ･･･ 出力の途中部分 ･･･
```

isubの実行結果のOUTを上書き更新していることにも注目

3. Lis逐次版のインストール(補足)

make installは実行しなくても利用可能:

```
$ make install
```

マニュアルには，最後に上記を実行するように記述されているが，これは，正式にシステムにインストールする(例えば，/usr/local/下など)場合に実行すれば良い．

この実習では，既に作成されている“test1 ”というプログラムを用いて説明するため，特にインストール先を指定しないでも良い．
現状で，生成済みの関連ファイル群は下記のディレクトリ構成である:

```
$HOME/lis-1.3.0_seq/ 下:
      include/
        lis_config.h, lis.h, lisf.h 他
      src/.libs/
        liblis.a  liblis.la@  liblis.lai
```

当実習では不要！

インストール先のデフォルトは /usr/local/ 下であるが，
それを変更する場合には，configure のステップまで戻って，下記のオプションを指定する．

```
$ ./configure --prefix={インストールディレクトリ}
```

configureで指定したTARGET=fujitsu_fx10_crossの実体

```
CC=fccpx FC=frtpx CFLAGS="-Kfast,ocl,preex" FCFLAGS="-Kfast,ocl,preex -Cpp
-fs" FCLDFLAGS="-mlcmain=main" ac_cv_sizeof_void_p=8 cross_compiling=yes
ax_f77_mangling="lower case, underscore, no extra underscore"
```

configureコマンドの引数として，これら全ての環境変数を上記のとおり指定したことと等価．
特に，赤文字で強調したコンパイラと cross_compiling=yes では，クロスコンパイリング環境であることにも注目．

4. isubコマンドでジョブ実行(逐次版)

isubコマンドによるジョブ実行1:

```
$ cd $HOME/lis-1.3.0_seq/test

$ isub -seq "./test1 testmat.mtx 0 sol1.txt rsd1.txt"
./test1 testmat.mtx 0 sol1.txt rsd1.txt
[INFO] PJM 0000 pjsub Job 437789 submitted.

$ lrt
-rw-r--r-- 1 t00003 gt00    3242 12月 13 18:08 2012 sol1.txt  # 解ベクトル1(solution)
-rw-r--r-- 1 t00003 gt00     208 12月 13 18:08 2012 rsd1.txt  # 相対残差1(residual)のノルム収束履歴
-rw-r--r-- 1 t00003 gt00     498 12月 13 18:08 2012 OUT       # std{out,err}ファイル

$ cat OUT
number of processes = 1
matrix size = 100 x 100 (460 nonzero entries)
initial vector x = 0
precision : double
solver    : BiCG 2
precon    : none
storage   : CSR
lis_solve : normal end

BiCG: number of iterations     = 15 (double = 15, quad = 0)
BiCG: elapsed time             = 4.179478e-04 sec.
BiCG:   preconditioner         = 5.769730e-05 sec.
BiCG:     matrix creation      = 3.099442e-05 sec.
BiCG:   linear solver          = 3.602505e-04 sec.
BiCG: relative residual 2-norm = 1.689581e-16
```

※「Lis逐次版のインストール(4/5)」のスライドと同じ出力
(make checkと同じコマンド，同じオプションを指定したということ)．

4. isubコマンドでジョブ実行(逐次版)

isubコマンドによるジョブ実行2:

```
$ isub -seq "./test1 testmat.mtx 0 sol2.txt rsd2.txt -i 3 -p 1"
./test1 testmat.mtx 0 sol2.txt res2.txt -i 3 -p 1
[INFO] PJM 0000 pjsub Job 437797 submitted.

$ lrt
-rw-r--r-- 1 t00003 gt00    3242 12月 13 18:08 2012 sol1.txt  # 解ベクトル1(solution)
-rw-r--r-- 1 t00003 gt00     208 12月 13 18:08 2012 rsd1.txt  # 相対残差1(residual)のノルム収束履歴
-rw-r--r-- 1 t00003 gt00    3242 12月 13 18:14 2012 sol2.txt  # 解ベクトル2(solution)
-rw-r--r-- 1 t00003 gt00     208 12月 13 18:14 2012 rsd2.txt  # 相対残差2(residual)のノルム収束履歴
-rw-r--r-- 1 t00003 gt00     493 12月 13 18:14 2012 OUT       # std{out,err}ファイル

$ cat OUT
number of processes = 1
matrix size = 100 x 100 (460 nonzero entries)
initial vector x = 0
precision : double
solver    : CGS 3
precon    : Jacobi
storage   : CSR
lis_solve : normal end

CGS: number of iterations     = 15 (double = 15, quad = 0)
CGS: elapsed time             = 1.056194e-03 sec.
CGS:   preconditioner         = 7.200241e-05 sec.
CGS:     matrix creation      = 3.099442e-05 sec.
CGS:   linear solver          = 9.841919e-04 sec.
CGS: relative residual 2-norm = 1.004890e-14
```

5. いま，Lisで何をやったのだろうか

```
$ isub  -seq  "./test1"     # 実行結果（OUTの内容）
Usage: test1 matrix_filename rhs_setting solution_filename residual_filename [options]

matrix_filename:    係数行列のファイルを指定
rhs_setting:        右辺項ベクトルファイル，あるいは，自動生成機能（主にアルゴリズムの性能評価用途）のIDを指定
                      0 : 拡張Matrix Market形式（右辺ベクトルが行列データに含まれている）を用いる
                      1 : b = (1,1,...,1)^T   を用いる
                      2 : b = A ×(1,1,...,1)^T   を用いる
solution_filename: 解ベクトルのファイル名
residual_filename: （アルゴリズム中のベクトルを用いた）相対残差ベクトルの2ノルムの履歴
[options]:          解法と前処理，その他のオプション指定

# 参考：マニュアル「テストプログラム」の「test1」小節
```

```
$ more testmat.mtx
%%MatrixMarket matrix coordinate real general
100 100 460 1 0
1 2 -1.0000000000000000000e+000        # 参考1：マニュアル 付録「ファイル形式」の
1 11 -1.0000000000000000000e+000         「拡張Matrix Market形式」小節
1 1  4.0000000000000000000e+000          testmat.mtx も係数行列と右辺項の両方の
2 3 -1.0000000000000000000e+000          データを格納した拡張形式．
2 12 -1.0000000000000000000e+000       # 参考2：
2 1 -1.0000000000000000000e+000          http://math.nist.gov/MatrixMarket/formats.html
2 2  4.0000000000000000000e+000
 …
```


5. 色々な解法で解いてみよう

解法 デフォルトはBiCG	選択オプション {a | n}：a，nどちらでも良い	補助オプション [ ]内はデフォルト値
CG	-i {cg | 1}	
BiCG	-i {bicg | 2}	
CGS	-i {cgs | 3}	
BiCGSTAB	-i {bicgstab | 4}	
BiCGSTAB(l)	-i {bicgstabl | 5}	-ell [2]　　解法の l の値
GPBiCG	-i {gpbicg | 6}	
TFQMR	-i {tfqmr | 7}	
Orthomin(m)	-i {orthomin | 8}	-restart [40]　リスタート m の値
GMRES(m)	-i {gmres | 9}	-restart [40]　リスタート m の値
Jacobi	-i {jacobi | 10}	
Gauss-Seidel	-i {gs | 11}	
SOR	-i {sor | 12}	-omega [1.9]　緩和係数ωの値(0<ω<2)

● 上記も含め，全22種類の解法が提供されている．

5. 色々な前処理を組合せてみよう

前処理 デフォルトは“なし”	選択オプション {a | n}：a，nどちらでも良い	補助オプション [ ]内はデフォルト値
なし	-p {none | 0}	
Jacobi	-p {jacobi | 1}	
ILU(k)	-p {ilu | 2}	-ilu_fill [0]　　フィルインレベル k
SSOR	-p {ssor | 3}	-ssor_w [1.0]　緩和係数ωの値(0<ω<2)
Hybrid	-p {hybrid | 4}	-hybrid_i [sor]　線型方程式系解法，他
I+S	-p {is | 5}	-is_alpha [1.0]　前処理中の変数α，他
SAINV	-p {sainv | 6}	-sainv_drop [0.05]　ドロップ基準
SA-AMG	-p {saamg | 7}	-saamg_unsym [false]　非対称版選択，他
Crout ILU	-p {iluc | 8}	-iluc_drop [0.05]　ドロップ基準，他
ILUT	-p {ilut | 9}	-ilut_drop [0.05]　ドロップ基準，他
Additive Schwarz	-adds true	-adds_iter [1]　　繰り返し回数

● 上記の補助オプションは一部のみ掲載している．詳細はマニュアル参照．

5. 色々なオプションを試してみよう

オプション [ ]内はデフォルト値	内容
-maxiter　[1000]	最大反復回数
-tol　[1.0e-12]	収束判定基準（相対残差ノルム）
-print　[0]	残差の画面表示
-scale　[0]	スケーリングの選択． ※ スケーリングされた係数行列，右辺項ベクトルは，元のデータに上書きされて，プログラムで実行される．
-initx_zeros　[true]	初期ベクトル x_0 の設定（デフォルトは全要素の値が 0）
-omp_num_threads　[t]	OMP実行スレッド数．t は最大スレッド数 *
-storage　[0]	行列格納形式
-storage_block　[2]	BSR，BSCのブロックサイズ

* FXでは，デフォルト設定にて FLIB_FASTOMP=TRUE となっているため，当オプションによるプログラムからのスレッド数指定ではエラーとなります．FXで当オプションを使用する場合，あらかじめ，環境変数にて「FLIB_FASTOMP=FALSE」「FLIB_CNTL_BARRIER_ERR=FALSE」を指定して下さい．

● 上記のオプションの内容の詳細はlisのマニュアル参照．

5. 色々なデータで実行してみよう

```
$ pwd
/home/t00003/lis-1.3.0_seq/test
$ ls  $HOME/sample/
library/  mat/  script/  setup.env  sh/

$ ln  -s  $HOME/sample/mat .
$ ls  -l  mat
lrwxrwxrwx 1 t00003 gt00 23 12月 13 18:38 2012 mat -> /home/t00003/sample/mat/
$ ls mat/
494_bus.mtx  add20_rhs1.mtx  add32_rhs1.mtx  memplus.mtx       nos3.rsa
494_bus.rsa  add32.mtx       add32_rhs2.mtx  memplus.rua
add20.mtx    add32.rua       fs_183_1.mtx    memplus_rhs1.mtx
add20.rua    add32_rhs1.dat  fs_183_1.rua    nos3.mtx
```

参考までに，matの後にディレクトリを示す「/」を付け忘れると：

```
$ ls mat       # これは，☆1のsetup.env 影響
mat@
```

- 拡張子
  mtx : Matrix Market形式
  rua, rsa : Harwell-Boeing形式
  dat : ベクトルのPLAIN形式
  (拡張子自体は任意)

```
$ more $HOME/sample/script/pjsub_seq.sh
 ・・・
#./test1 ./mat/add32.mtx 2 sol.txt res.txt -i 5 -p 2
#./test1 ./mat/add32.mtx ./mat/add32_rhs1.dat sol.txt res.txt -i 5 -p 2
#./test1 ./mat/add32.rua 2 sol.txt res.txt -i 5 -p 2
#./test1 ./mat/nos3.mtx 2 sol.txt res.txt -i 5 -p 2
 ・・・
```

- 各々をバッチジョブ実行するときは，左端冒頭の「#」(コメントアウト)記号を削除

6. 入力データ形式について

- Matrix Market(MM)形式は，データの圧縮度は低いものの，行列のインデックスと値との対応を確認し易く，また，多くのアプリケーションでもMM形式をサポートしたりツール提供されているので，初級者や中級者にとっては都合が良いと思われる．
- Lisのtest1.cでは，MM形式のデータを読込み，CSR形式に変換する．MPIのデータ分散にも対応している．
- 対称行列の場合には，対角要素から下三角の要素のみが格納されている．この場合，ヘッダには，半角スペースに続いて「symmetric」の文字列が記載されている．
  → mat/下の「494_bus.mtx」「nos3.mtx」が該当
- 非対称行列の場合には，「general」の文字列が記載されている．
  → mat/下の「add{20,32}.mtx」「fs_183_1.mtx」「memplus.mtx」が該当

```
add32.mtx       : MMサイトで提供されているMM形式の係数行列データ
add32_rhs1.mtx  :       〃              MM形式の右辺項ベクトルデータ
add32.rua       :       〃              Harwell-Boeing(HB)形式の係数行列データ
```

MMサイトで提供されている add32_rhs1.mtx のヘッダとフォーマットでは，Lisで読み取れないので，当実習では以下の2通りの修正データを用意してます．問題点と修正内容は，各自で確認して下さい．
add20，memplus等も同様ですが，サンプルではadd32のみ修正データを用意しています．

```
add32_rhs1.dat : add32_rhs1.mtxを元にPLAIN形式にしたファイル
add32_rhs2.mtx : Lisで読み取り可能なベクトル用拡張MM形式のデータ
```


7. Lis並列版の準備

逐次版と同様，ディレクトリごとコピーして並列版の環境を構築する：

```
$ cd  $HOME/lis-1.3.0/test/              # lis-1.3.0_seq ではないことに注意

$ ln  -s  $HOME/sample/mat .             # matをシンボリックリンク

$ ls -l mat                              # シンボリックリンクの確認
lrwxrwxrwx 1 t00003 gt00 23 12月 13 18:42 2012 mat -> /home/t00003/sample/mat/

$ cd     (または，「 cd ~ 」や「 cd $HOME 」など)

$ cp -rp  lis-1.3.0  lis-1.3.0_omp       # LisのOpenMP(omp)版の環境準備
$ cp -rp  lis-1.3.0  lis-1.3.0_mpi       # LisのMPI(mpi)版の環境準備

$ cp -rp  lis-1.3.0  lis-1.3.0_hyb       # LisのHybrid(mpi+omp)版の環境準備


$ cd  lis-1.3.0_hyb
```


7. Lis並列(Hybrid)版のインストール(1/3)

Lis並列(Hybrid)版のconfigure：

```
$ pwd
/home/t00003/lis-1.3.0_hyb

$ ./configure --enable-omp --enable-mpi --enable-saamg TARGET=fujitsu_fx10_cross
              # ここで omp と mpi の両方を指定する(もちろん，片方だけ指定も可能)
  ・・・ 途中省略 ・・・

  ・・・ 出力の最後部 ・・・
Enable Fortran Interface       = no
Enable SA-AMG Preconditioner   = yes
Enable MPI                     = yes
Enable OpenMP                  = yes
Enable Quad Precision          = no
Enable Long Long Int           = no
Enable Dynamic Linking         = no
Enable Profiling               = no

C Compiler         = mpifccpx
C Flags            = -Kfast,ocl,preex -w -DUSE_MPI -Kopenmp -DHAVE_CONFIG_H
C Libraries        = -lm
F90 Compiler       = mpifrtpx
F90 Flags          = -Kfast,ocl,preex -Cpp -fs -DZERO_ORIGIN=1 -DUSE_MPI -Kopenmp
F90 Libraries      =
MPI Libraries      =
MPIRUN Script      = mpiexec
```

赤文字の8項目をチェック

- configure実行後は，作業ディレクトリやパスを変更しないように(生成されるMakefile中では絶対パスが用いられている)．

7. Lis並列(Hybrid)版のインストール(2/3)

Lis並列(Hybrid)版のmake:

```
$ make

  ･･･  途中省略  ･･･

  ･･･ 出力の最後部 ･･･

/bin/sh ../libtool --mode=link mpifrtpx  -Kopenmp -mlcmain=main -o etest6 -lm   etest6.o -
L../src -llis -lm
libtool: link: mpifrtpx -Kopenmp -mlcmain=main -o etest6 etest6.o  -L/home/t00003/lis-
1.3.0_hyb/src /home/t00003/lis-1.3.0_hyb/src/.libs/liblis.a -lm
make[1]: ディレクトリ `/home/t00003/lis-1.3.0_hyb/test' から出ます
make[1]: ディレクトリ `/home/t00003/lis-1.3.0_hyb' に入ります
make[1]: `all-am' に対して行うべき事はありません.
make[1]: ディレクトリ `/home/t00003/lis-1.3.0_hyb' から出ます
```

最後部でErrorが出て
いなければOK

7. Lis並列(Hybrid)版のインストール(3/3)

Lis並列(Hybrid)版のインストールチェック(make check):

```
$ isub -hyb "make check"
```

#Lisのmake checkでは，MPIプログラムであっても，mpiexecコマンドは不要
(make checkで呼び出すtest/test.sh 内でmpiexecを記述しているため)

```
$ cat OUT
  ･･･ 出力の途中部分 ･･･
make  check-TESTS
make[2]: Entering directory `/home/t00003/lis-1.3.0_hyb/test'
=== Running test test.sh

checking linear solvers...

number of processes = 2
max number of threads = 16
number of threads = 2
matrix size = 100 x 100 (460 nonzero entries)
initial vector x = 0
precision : double
solver    : BiCG 2
precon    : none
storage   : CSR
lis_solve : normal end

BiCG: number of iterations     = 15 (double = 15, quad = 0)
BiCG: elapsed time             = 2.397419e-03 sec.
BiCG:   preconditioner         = 1.086006e-04 sec.
BiCG:     matrix creation      = 2.899906e-06 sec.
BiCG:   linear solver          = 2.288818e-03 sec.
BiCG: relative residual 2-norm = 1.351335e-16

 ･･･
```

#Lisの make check では，
ノード数(number of processes)=2 と
スレッド数(number of threads)=2
は固定されている.

赤文字の項目をチェック

下線の項目は計測時間なので，
一致して無くても良い

7. isubコマンドでジョブ実行(Hybrid版)

isubコマンドによるジョブ実行:

```
$ cd $HOME/lis-1.3.0_hyb/test
```

#Hybrid版はMPIプログラムでもあるので，mpiexecコマンドが必要

```
$ isub -hyb "mpiexec ./test1 testmat.mtx 0 sol1.txt rsd1.txt"
mpiexec ./test1 testmat.mtx 0 sol1.txt rsd1.txt
[INFO] PJM 0000 pjsub Job 437872 submitted.

$ lrt
-rw-r--r-- 1 t00003 gt00    3242 12月 13 19:00 2012 sol1.txt  # 解ベクトル1(solution)
-rw-r--r-- 1 t00003 gt00     208 12月 13 19:00 2012 rsd1.txt  # 相対残差1(residual)のノルム収束履歴
-rw-r--r-- 1 t00003 gt00     547 12月 13 19:00 2012 OUT       # std{out,err}ファイル

$ cat OUT
number of processes = 4
max number of threads = 16
number of threads = 8
matrix size = 100 x 100 (460 nonzero entries)
initial vector x = 0
precision : double
solver    : BiCG 2
precon    : none
storage   : CSR
lis_solve : normal end

BiCG: number of iterations     = 15 (double = 15, quad = 0)
BiCG: elapsed time             = 3.740030e-03 sec.
BiCG:   preconditioner         = 1.248000e-04 sec.
BiCG:     matrix creation      = 2.699904e-06 sec.
BiCG:   linear solver          = 3.615230e-03 sec.
BiCG: relative residual 2-norm = 2.840829e-16
```

isubコマンドのデフォルトでは，
ノード数(number of processes)=4
スレッド数(number of threads)=8
と指定している.
「0.FXでの実習環境設定(5/5)」参照

7. isubコマンドでジョブ実行(Hybrid版)

isubコマンドにてpjsubのオプション指定したジョブ実行:

```
$ isub -hyb "mpiexec ./test1 testmat.mtx 0 sol2.txt rsd2.txt" -o OUT1 -L "node=2"
mpiexec ./test1 testmat.mtx 0 sol2.txt rsd2.txt
[INFO] PJM 0000 pjsub Job 438648 submitted.

$ pjstat
JOB_ID  JOB_NAME   STATUS  PROJECT RSCGROUP START_DATE       ELAPSE    TOKEN NODE:COORD
438648  isub       RUNNING gt00    lecture  (12/14 15:18:31) 00:00:00    0.0 2

$ lrt
-rw-r--r-- 1 t00003 gt00    3242 12月 13 19:00 2012 sol1.txt  # 解ベクトル1(solution)
-rw-r--r-- 1 t00003 gt00     208 12月 13 19:00 2012 rsd1.txt  # 相対残差1(residual)のノルム収束履歴
-rw-r--r-- 1 t00003 gt00     547 12月 13 19:00 2012 OUT       # std{out,err}ファイル
-rw-r--r-- 1 t00003 gt00    3242 12月 14 15:18 2012 sol2.txt  # 解ベクトル2(solution)
-rw-r--r-- 1 t00003 gt00     208 12月 14 15:18 2012 rsd2.txt  # 相対残差2(residual)のノルム収束履歴
-rw-r--r-- 1 t00003 gt00     547 12月 14 15:18 2012 OUT1      # std{out,err}ファイル2

$ diff OUT OUT1
2c2
< number of processes = 4
---
> number of processes = 2
14,18c14,18
< BiCG: elapsed time             = 3.740030e-03 sec.
< BiCG:   preconditioner         = 1.248000e-04 sec.
< BiCG:     matrix creation      = 2.699904e-06 sec.
< BiCG:   linear solver          = 3.615230e-03 sec.
< BiCG: relative residual 2-norm = 2.840829e-16
---
  ･･･ 以下省略 ･･･
```

● pjsubオプションの指定は，(isubに限らず一般的なpjsub用の)
ジョブスクリプト内での指定よりも，pjsubコマンドの引数(option)
指定の方が優先される.

isubコマンドのデフォルトでは，
・出力ファイル: OUT
・ノード数(number of processes)=4
であるが，今回は pjsubコマンドのオプション
として -o OUT1 -L "node=2" を指定した.

7. isubコマンドでジョブ実行(Hybrid版)

isubコマンドにてpjsubのオプション指定したジョブ実行：

```
$ isub -hyb -th 4 "mpiexec ./test1 testmat.mtx 0 sol3.txt rsd3.txt" -o OUT2

$ lrt
-rw-r--r-- 1 t00003 gt00    3242 12月 13 19:00 2012 sol1.txt
-rw-r--r-- 1 t00003 gt00     208 12月 13 19:00 2012 rsd1.txt
-rw-r--r-- 1 t00003 gt00     547 12月 13 19:00 2012 OUT
-rw-r--r-- 1 t00003 gt00    3242 12月 14 15:18 2012 sol2.txt
-rw-r--r-- 1 t00003 gt00     208 12月 14 15:18 2012 rsd2.txt
-rw-r--r-- 1 t00003 gt00     547 12月 14 15:18 2012 OUT1
-rw-r--r-- 1 t00003 gt00    3242 12月 16 13:46 2012 sol3.txt  # 解ベクトル3(solution)
-rw-r--r-- 1 t00003 gt00     208 12月 16 13:46 2012 rsd3.txt  # 相対残差3(residual)のノルム収束履歴
-rw-r--r-- 1 t00003 gt00     547 12月 16 13:46 2012 OUT2      # std{out,err}ファイル3

$ diff OUT OUT2
4c4
< number of threads = 8
---
> number of threads = 4
14,18c14,18
< BiCG: elapsed time             = 3.740030e-03 sec.
< BiCG:   preconditioner         = 1.248000e-04 sec.
< BiCG:     matrix creation      = 2.699904e-06 sec.
< BiCG:   linear solver          = 3.615230e-03 sec.
< BiCG: relative residual 2-norm = 2.840829e-16
---
  ･･･ 以下省略 ･･･
```

isubコマンドのデフォルトでは，

・ノード数(number of threads)=8

であるが，今回は pjsubコマンドのオプション として -th 4 を指定した．

7. Lisのその他の並列(OMP)版(1/2)

OMP版： configureオプションとisubコマンドの引数(-omp)に注目：

```
$ cd $HOME/lis-1.3.0_omp
$ ./configure --enable-omp --enable-saamg TARGET=fujitsu_fx10_cross
  Enable Fortran Interface       = no
  Enable SA-AMG Preconditioner   = yes
  Enable MPI                     = no
  Enable OpenMP                  = yes
  Enable Quad Precision          = no
  Enable Long Long Int           = no
  Enable Dynamic Linking         = no
  Enable Profiling               = no

  C Compiler         = fccpx
  C Flags            = -Kfast,ocl,preex -w -Kopenmp -DHAVE_CONFIG_H
  C Libraries        = -lm
  F90 Compiler       = frtpx
  F90 Flags          = -Kfast,ocl,preex -Cpp -fs -DZERO_ORIGIN=1 -Kopenmp
  F90 Libraries      =

$ make
$ isub -omp "make check"
number of processes = 1
max number of threads = 16
number of threads = 2
matrix size = 100 x 100 (460 nonzero entries)
   ･･･ 中略 ･･･
storage    : CSR
lis_solve : normal end
```

#詳細な確認は
次のスライド(2/2)を参照

7. Lisのその他の並列(OMP)版(2/2)

OMP版： isubコマンドの引数(-omp)とOUTの内容に注目：

```
$ cd $HOME/lis-1.3.0_omp/test
$ isub -omp "./test1 testmat.mtx 0 sol1.txt rsd1.txt"

$ cat OUT
number of processes = 1
max number of threads = 16
number of threads = 8
matrix size = 100 x 100 (460 nonzero entries)
initial vector x = 0
precision : double
solver     : BiCG 2
precon     : none
storage    : CSR
lis_solve : normal end

BiCG: number of iterations     = 15 (double = 15, quad = 0)
BiCG: elapsed time             = 2.825022e-03 sec.
BiCG:   preconditioner         = 1.111031e-04 sec.
BiCG:     matrix creation      = 9.536743e-07 sec.
BiCG:   linear solver          = 2.713919e-03 sec.
BiCG: relative residual 2-norm = 1.533189e-16
```


7. Lisのその他の並列(MPI)版(1/2)

MPI版： configureオプション，isubコマンドの引数(-mpi)，OUTの内容に注目：

```
$ cd $HOME/lis-1.3.0_mpi
$ ./configure --enable-mpi --enable-saamg TARGET=fujitsu_fx10_cross
  Enable Fortran Interface       = no
  Enable SA-AMG Preconditioner   = yes
  Enable MPI                     = yes
  Enable OpenMP                  =
  Enable Quad Precision          = no
  Enable Long Long Int           = no
  Enable Dynamic Linking         = no
  Enable Profiling               = no

  C Compiler         = mpifccpx
  C Flags            = -Kfast,ocl,preex -w -DUSE_MPI -DHAVE_CONFIG_H
  C Libraries        = -lm
  F90 Compiler       = mpifrtpx
  F90 Flags          = -Kfast,ocl,preex -Cpp -fs -DZERO_ORIGIN=1 -DUSE_MPI
  F90 Libraries      =
  MPI Libraries      =
  MPIRUN Script      = mpiexec

$ make
$ isub -mpi "make check"
number of processes = 2
matrix size = 100 x 100 (460 nonzero entries)
   ･･･ 中略 ･･･
storage    : CSR
lis_solve : normal end
```

#詳細な確認は
次のスライド(2/2)を参照

7. Lisのその他の並列(MPI)版(2/2)

MPI版: isubコマンドの引数(-mpi), OUTの内容に注目:

```
$ cd $HOME/lis-1.3.0_mpi/test
$ isub -mpi "mpiexec ./test1 testmat.mtx 0 sol1.txt rsd1.txt"

$ cat OUT
number of processes = 4
matrix size = 100 x 100 (460 nonzero entries)
initial vector x = 0
precision : double
solver    : BiCG 2
precon    : none
storage   : CSR
lis_solve : normal end

BiCG: number of iterations     = 15 (double = 15, quad = 0)
BiCG: elapsed time             = 2.942821e-03 sec.
BiCG:   preconditioner         = 3.380026e-05 sec.
BiCG:     matrix creation      = 2.900138e-06 sec.
BiCG:   linear solver          = 2.909021e-03 sec.
BiCG: relative residual 2-norm = 1.550156e-16
```


8. test1 処理フローの概要(1)

```
LIS_INT main(LIS_INT argc, char* argv[] ) {
  lis_initialize

  lis_matrix_create                  係数行列 A をmalloc
  lis_vector_create                  ベクトル b, x をmalloc
  lis_input(A,b,x,argv[1]);          A(MMかHB形式), b(ファイル入力の場合)
                                     , x(初期値)を読込み

  lis_matrix_set_type                行列データをCSRへ変換
  lis_matrix_convert                   〃
  右辺項 b の作成処理

  lis_solve(A,b,x,solver);           求解ルーチン

  lis_solver_get_timeex              タイマーの値をget
  lis_solver_get_residualnorm        真の残差ノルム算出

  lis_output_vector(x,LIS_FMT_MM,argv[3]);   解を出力
  lis_solver_output_rhistory(solver, argv[4]); 残差履歴を出力

  lis_finalize();
  return 0;
}
```

MMとHB以外の形式は，マニュアル5節「行列格納形式」のサンプルや，特にCSRを扱った例としてtest[2-4].cも参照

各関数の引数は，実際のtest1.cファイルの記述を確認して下さい．

8. test1 処理フローの概要(2)

```
int lis_solve( ) {
  lis_precon_create                  前処理演算の設定
  lis_solve_kernel(A, b, x, solver, precon);   求解のための事前処理
}
```

```
int lis_solve_kernel( ) {
  test1の引数をチェック

  x0を設定

  if(I+S前処理選択) スケーリング処理        I+S前処理は要スケーリング
  else (スケーリング選択) スケーリング処理   CG法ではSYMM_DIAG
                                  ※1 scalingでは A, b のデータを上書き

  lis_solver_execute                 求解(例えば，lis_gmresを呼び出す)

  y=D^{-1/2}x          「SYMM_DIAG(対角scaling)」&&「I+S前処理以外」のとき実行

  lis_vector_nrm2(t,&nrm2);          真の残差ベクトルの2ノルム(絶対残差)
                                  ※2 scaling後の系に対して算出(cf. ※1)
}
```


8. プログラム内で係数行列作成する場合：Lisのtest4,5の例

$$Ax = b \quad b = Ax, \quad x = [1.0, 1.0, \cdots, 1.0]^T$$

#MPIプログラム(Hybridも)の場合，mpiexecコマンドの引数として実行

```
$ ./test4 [options]
```

$$A = \begin{bmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & 2 \end{bmatrix}$$

係数行列Aの次数n=12(プログラム内で定義)の3重対角行列

```
$ ./test5 n gamma [options]
```

$$A = \begin{bmatrix} 2 & 1 & & & 0 \\ 0 & 2 & 1 & & \\ \gamma & 0 & 2 & 1 & \\ & \ddots & \ddots & \ddots & \ddots \\ & & \gamma & 0 & 2 & 1 \\ 0 & & & \gamma & 0 & 2 \end{bmatrix}$$

係数行列Aの次数nのToeplitz行列

8. test4 係数行列作成の概要

```
LIS_INT main(LIS_INT argc, char* argv[] ) {
  lis_initialize

  lis_matrix_create(LIS_COMM_WORLD,&A);     係数行列 A をmalloc
  lis_matrix_set_size(A,0,n);               A のglobalな行数サイズ（n）を設定
  lis_matrix_get_size(A,&n,&gn);            A のlocal size(n)とglobal size(gn)情報を取得
  lis_matrix_get_range(A,&is,&ie);          分割後のAの当該プロセッサ内での領域を与える

  for(i=is;i<ie;i++)                        test4 では配列Aに値を設定して係数行列を作成
  {
    if( i>0   )   lis_matrix_set_value(LIS_INS_VALUE,i,i-1,-1.0,A);    i, j=i-1 に -1.0を設定
    if( i<gn-1 ) lis_matrix_set_value(LIS_INS_VALUE,i,i+1,-1.0,A);   i, j=i+1 に -1.0を設定
    lis_matrix_set_value(LIS_INS_VALUE,i,i,2.0,A);                    i, j=i に 2.0を設定
  }
  lis_matrix_set_type(A,LIS_MATRIX_CSR);    行列Aの格納形式をCSRとして設定する
  lis_matrix_assemble(A);                   行列Aをプログラム内で使用可能にする

  ・・・ 途中省略 ・・・

  lis_finalize();
  return 0;
}
```


8. test5 係数行列作成の概要

```
LIS_INT main(LIS_INT argc, char* argv[] ) {
  lis_initialize

  gn  = atoi(argv[1]);                      1番目の引数にて行列サイズを設定
  gamma  = atof(argv[2]);                   2番目の引数にてガンマの値を設定

  lis_matrix_create(LIS_COMM_WORLD,&A);     係数行列 A をmalloc
  lis_matrix_set_size(A,0,gn);              A のglobalな行数サイズ（gn）を設定
  lis_matrix_get_size(A,&n,&gn);            A のlocal size(n)とglobal size(gn)情報を取得
  lis_matrix_malloc_csr(n,3*n,&ptr,&index,&value);
  lis_matrix_get_range(A,&is,&ie);          分割後のAの当該プロセッサ内での領域を与える
  k = 0;
  ptr[0] = 0;
  for(ii=is;ii<ie;ii++)                     test5 では，直接，CSR形式としてデータ格納
  {
    if( ii>1 )   { jj = ii - 2; index[k] = jj; value[k++] = gamma;}
    if( ii<gn-1 ) { jj = ii + 1; index[k] = jj; value[k++] = 1.0;}
    index[k] = ii; value[k++] = 2.0;
    ptr[ii-is+1] = k;
  }
  lis_matrix_set_csr(ptr[ie-is],ptr,index,value,A);  CSR形式の配列を行列Aに関連付ける
  lis_matrix_assemble(A);                   行列Aをプログラム内で使用可能にする
  ・・・ 以下省略 ・・・
```

講習内容

前半の内容

0. 準備，FXでの実習環境設定
1. Lisの概要
2. Lisインストールの準備作業
3. Lis逐次版のインストール
4. ジョブ実行（逐次版）
5. Lis使用に関する解説
6. 入力データ形式について
7. Lis並列版のインストール・実行
8. Lisテストプログラムの概要

後半の内容

9. PETScの概要
10. FXでのPETSc利用の準備
11. FXでのPETSc-逐次版利用例
12. FXでのPETSc-MPI版利用例
13. PETSc-複素数演算例題
14. PETSc使用に関する解説
15. PETScテストプログラムの概要
16. KSPに関する盲点
17. 両ライブラリの習得要領

9. PETScの概要（線形方程式版）

[http://www.mcs.anl.gov/petsc/]

PETSc (Portable, Extensible Toolkit for Scientific Computation)

- 開発元：ANL : Argonne National Laboratory
  異なるバージョン間の非互換が大きいので注意.
  インストール（特にconfigure）は少々難.
- 線形方程式求解アルゴリズムの逐次版，並列版（MPIのみ）
  14 反復解法 × 10 前処理（“前処理無し”も含める） ＋ 2 直接解法
- 複数のデータ形式をサポート：
  AIJ(CSR形式のこと), Block AIJ, Symmetric Block AIJ, Dense, Matrix-Free

Component of PETSc: [PETSc-3.3-p4 ver.]

反 復 解 法
非定常反復解法
Richardson, Chebychev, CG, BiCG, GMRES, FGMRES, Deflated GMRES, GCR, BiCGStab, CGS, TFQMR(1), TFQMR(2), CR, LSQR

前 処 理
None, (Point) Jacobi, Block Jacobi, SOR (and SSOR), SOR with Eisenstat trick, IC, ILU, Additive Schwarz, Linear solver, Combination of preconditioner

直接解法
LU, Cholesky

10. FXでのPETSc利用の準備(1/2)

PETSc実習環境の準備：

Oakleaf-fx のPETScインストールディレクトリ：
/usr/local/PETSc/3.3-p4/

```
$ pwd
/home/t00003

$ cp -rp ~/sample/library/petsc-3.3-p4_samp  .
$ cd petsc-3.3-p4_samp
$ ls
ex1.c  ex11.c@  ex11_mod.c  ex11_org.c  ex1_mod.c  ex23.c  makefile
```

#ex1_mod.c と ex11_mod.c 以外は全て oakleaf-fx の
/usr/local/PETSc/3.3-p4/src/ksp/ksp/examples/tutorials/
の下のファイルをコピーしたもの(ex11_org.cのみ，ファイル名変更している).

```
$ ls -l ex11.c
lrwxrwxrwx 1 t00003 gt00 10  2月 19 16:08 2013 ex11.c -> ex11_mod.c
```

#ex11.cは ex11_mod.cの方を使用する.
PETSc提供のオリジナルex11.c(ex11_org.cのこと)は不完全なコード
(FX固有の問題では無い).

10. FXでのPETSc利用の準備(2/2)

PETSc実習環境の準備：

```
$ module list
Currently Loaded Modulefiles:
  1) TCSuite/GM-1.2.1-04
```

#テキスト作成後にTCSuiteの版数が上がっている場合があるが，
当講習会の範囲では，特に気にしないで良い（見込み）.

```
$ module avail PETSc

------------------- /usr/share/Modules/modulefiles/apps --------------------
PETSc/3.0.0                    PETSc-complex/3.3-p4(default)
PETSc/3.3-p4(default)

$ module load PETSc  （「 $ module load PETSc/3.3-p4 」でも可 ）
$ module list
Currently Loaded Modulefiles:
  1) TCSuite/GM-1.2.1-04   2) PETSc/3.3-p4

$ env | grep -i petsc
PETSC_ARCH=linux
LIBRARY_PATH=/usr/local/PETSc/3.3-p4/lib
FORT90CPX=-I/usr/local/PETSc/3.3-p4/include/finclude -L/usr/local/PETSc/3.3-p4/lib
INCLUDE_PATH=/usr/local/PETSc/3.3-p4/include
FCCpx_ENV=-I/usr/local/PETSc/3.3-p4/include -L/usr/local/PETSc/3.3-p4/lib
PWD=/home/t00003/petsc-3.3-p4_samp
_LMFILES_=/usr/share/Modules/modulefiles/utils/TCSuite/GM-1.2.1-
03:/usr/share/Modules/modulefiles/apps/PETSc/3.3-p4
LOADEDMODULES=TCSuite/GM-1.2.1-03:PETSc/3.3-p4
fccpx_ENV=-I/usr/local/PETSc/3.3-p4/include -L/usr/local/PETSc/3.3-p4/lib
PETSC_DIR=/usr/local/PETSc/3.3-p4
```

11. FXでのPETSc-逐次版利用例(1/3)

ex1.c ： PETScの逐次版サンプルプログラム

```
$ module list
Currently Loaded Modulefiles:
  1) TCSuite/GM-1.2.1-04    2) PETSc/3.3-p4

$ pwd
/home/t00003/petsc-3.3-p4_samp

$ make ex1

  ･･･ makeのメッセージ出力 ･･･

#途中で下記のメッセージが表示されるが，特に気にしないでも良い(見込み)．
  oakleaf-fx のバージョンが古いということだが，当実習内での不都合は無い(見込み)．
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
The version of PETSc you are using is out-of-date, we recommend updating to the new
release
 Available Version: 3.3.5    Installed Version: 3.3.4
http://www.mcs.anl.gov/petsc/download/index.html
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 ･･･ 以下省略 ･･･
```


11. FXでのPETSc-逐次版利用例(2/3)

ex1.c ： PETScの逐次版サンプルプログラム

```
$ isub  -seq  "mpiexec ./ex1"
 または
$ isub  -mpi  "mpiexec -n 1 ./ex1"
```

ex1は逐次版であるが，MPIコードの1ノード用としてコーディングされている．したがって，isubコマンドでもMPIの1ノードとして実行する．

```
$ cat OUT
KSP Object: 1 MPI processes
  type: gmres
    GMRES: restart=30, using Classical (unmodified) Gram-Schmidt
Orthogonalization with no iterative refinement
    GMRES: happy breakdown tolerance 1e-30
  maximum iterations=10000, initial guess is zero
  tolerances:  relative=1e-05, absolute=1e-50, divergence=10000
  left preconditioning
  using PRECONDITIONED norm type for convergence test
PC Object: 1 MPI processes
  type: jacobi
  linear system matrix = precond matrix:
  Matrix Object:    1 MPI processes
    type: seqaij
    rows=10, cols=10
    total: nonzeros=28, allocated nonzeros=50
    total number of mallocs used during MatSetValues calls =0
      not using I-node routines
```


11. FXでのPETSc-逐次版利用例(3/3)

ex1.c ： PETScの逐次版サンプルプログラム

```
$ isub -seq "mpiexec ./ex1 -pc_type sor -ksp_type cgs"

$ cat OUT
KSP Object: 1 MPI processes
  type: cgs
  maximum iterations=10000, initial guess is zero
  tolerances:  relative=1e-05, absolute=1e-50, divergence=10000
  left preconditioning
  using PRECONDITIONED norm type for convergence test
PC Object: 1 MPI processes
  type: sor
    SOR: type = local_symmetric, iterations = 1, local iterations = 1, omega
= 1
  linear system matrix = precond matrix:
  Matrix Object:    1 MPI processes
    type: seqaij
    rows=10, cols=10
    total: nonzeros=28, allocated nonzeros=50
    total number of mallocs used during MatSetValues calls =0
      not using I-node routines
Norm of error 1.57716e-06, Iterations 6

$ isub -seq "mpiexec ./ex1 -help"    # サンプルコードでは，このようにしてhelpを参照できる
```


12. FXでのPETSc-MPI版利用例(1/2)

ex23.c ： PETScのMPI版サンプルプログラム

```
$ ls
OUT  ex1*  ex11.c@  ex11_mod.c  ex11_org.c  ex11.c  ex1_mod.c  ex23.c  makefile
$ module list
Currently Loaded Modulefiles:
  1) TCSuite/GM-1.2.1-04    2) PETSc/3.3-p4
$ make ex23
 ･･･ 省略 ･･･
$ isub -mpi "mpiexec ./ex23"
$ cat OUT
```

#実習状況により，表示されるファイル群は異なりますが，少なくとも赤文字のファイルは必須．

ex23はMPI並列版であるので，複数ノードで実行．

```
KSP Object: 4 MPI processes
  type: gmres
    GMRES: restart=30, using Classical (unmodified) Gram-Schmidt Orthogonalization with
no iterative refinement
    GMRES: happy breakdown tolerance 1e-30
  maximum iterations=10000, initial guess is zero
  tolerances:  relative=1e-07, absolute=1e-50, divergence=10000
  left preconditioning
  using PRECONDITIONED norm type for convergence test
PC Object: 4 MPI processes
  type: jacobi
  linear system matrix = precond matrix:
  Matrix Object:    4 MPI processes
    type: mpiaij
    rows=10, cols=10
    total: nonzeros=28, allocated nonzeros=70
    total number of mallocs used during MatSetValues calls =0
      not using I-node (on process 0) routines
```

12. FXでのPETSc-MPI版利用例(2/2)

ex23.c : PETScのMPI版サンプルプログラム

```
$ isub -mpi "mpiexec ./ex23 -pc_type sor -ksp_type cgs"
$ cat OUT
KSP Object: 4 MPI processes
  type: cgs
  maximum iterations=10000, initial guess is zero
  tolerances:  relative=1e-07, absolute=1e-50, divergence=10000
  left preconditioning
  using PRECONDITIONED norm type for convergence test
PC Object: 4 MPI processes
  type: sor
    SOR: type = local_symmetric, iterations = 1, local iterations = 1, omega
= 1
  linear system matrix = precond matrix:
  Matrix Object:   4 MPI processes
    type: mpiaij
    rows=10, cols=10
    total: nonzeros=28, allocated nonzeros=70
    total number of mallocs used during MatSetValues calls =0
      not using I-node (on process 0) routines
```


13. PETSc-複素数演算例題(MPI版1/2)

ex11.c : PETScの複素数演算サンプルプログラム(MPI版)

```
$ ls
OUT   ex1.c    ex11_mod.c  ex1_mod.c  ex23.c      #実習状況により，表示されるファイル群は異なり
ex1*  ex11.c@  ex11_org.c  ex23*      makefile     ますが，少なくとも赤文字のファイルは必須．

$ ls -l ex11.c
lrwxrwxrwx 1 t00003 gt00 10  2月 19 16:08 2013 ex11.c -> ex11_mod.c

$ module list                                  #PETSc提供のオリジナル ex11.c はコードが不完全のため，
Currently Loaded Modulefiles:                  実行時にアペンドする．ex11_mod.cは伊藤が修正したもの．
  1) TCSuite/GM-1.2.1-04   2) PETSc/3.3-p4

$ module unload PETSc
$ module list
Currently Loaded Modulefiles:
  1) TCSuite/GM-1.2.1-04

$ module load PETSc-complex
$ module list
Currently Loaded Modulefiles:
  1) TCSuite/GM-1.2.1-04    2) PETSc-complex/3.3-p4

$ make ex11
$ isub -mpi "mpiexec ./ex11"
```

PETScで複素数演算を行うために，complex版に切替える．
実数演算でもcomplex版（PETScインストールのcofigureにてcomplexオプションを有効にしている）で良いのでは無いかと思うのだが，詳細は未確認．

13. PETSc-複素数演算例題(MPI版2/2)

ex11.c : PETScの複素数演算サンプルプログラム(MPI版)

```
$ isub -mpi "mpiexec ./ex11"
$ cat OUT
1+1i
Norm of error 0.00050668 iterations 23
1+1i
1+1i
1+1i


$ isub -mpi "mpiexec ./ex11 -pc_type sor -ksp_type cgs"
$ cat OUT
1+1i
Norm of error 8.56439e-05 iterations 28
1+1i
1+1i
1+1i
```


14. 色々な解法で解いてみよう

解法 デフォルトはGMRES	KSPType	Options Database Name (-ksp_typeの引数)
Richardson	KSPRICHARDSON	richardson
Chebychev	KSPCHEBYCHEV	chebychev
Conjugate Gradient	KSPCG	cg
BiConjugate Gradient	KSPBICG	bicg
Generalized Minimal Residual	KSPGMRES	gmres
BiCGSTAB	KSPBCGS	bcgs
Conjugate Gradient Squared	KSPCGS	cgs
Transpose-Free Quasi-Minimal Residual (1)	KSPTFQMR	tfqmr
Transpose-Free Quasi-Minimal Residual (2)	KSPTCQMR	tcqmr
Conjugate Residual	KSPCR	cr
Least Squares Method	KSPLSQR	lsqr
Shell for no KSP method	KSPPREONLY	preonly

● 上記は3.0.0版のuser manualの一覧であるが，他の解法も利用可能（3.3-p4版のuser manual参照）．

14. 色々な前処理を組合せてみよう

前処理 デフォルトはJacobi	PCType	Options Database Name (-pc_typeの引数)
Jacobi	PCJACOBI	jacobi
Block Jacobi	PCBJACOBI	bjacobi
SOR (and SSOR)	PCSOR	sor
SOR with Eisenstat trick	PCEISENSTAT	eisenstat
Incomplete Cholesky	PCICC	icc
Incomplete LU	PCILU	ilu
Additive Schwarz	PCASM	asm
Linear solver	PCKSP	ksp
Combination of preconditioners	PCCOMPOSITE	composite
LU	PCLU	lu
Cholesky	PCCholesky	cholesky
No preconditioning	PCNONE	none

● 上記はuser manualの記述であるが，ソースコードを眺めると他の前処理も利用できそうな様子である．

14. 色々なオプションを試してみよう

主なオプション [ ]内はデフォルト値	内容
-ksp_max_it [100000]	最大反復回数
-ksp_rtol [1.0e-5]	収束判定基準(相対残差ノルム) $\|r_k\|_2 < \max(rtol \times \|b\|_2, atol)$
-ksp_atol [1.0e-50]	収束判定基準(絶対残差ノルム)
-ksp_divtol [1.0e5]	発散状況の判定基準(相対残差ノルム) $\|r_k\|_2 > dtol \times \|b\|_2$
-ksp_right_pc [left]	アルゴリズムの前処理系(デフォルトは左前処理系)
-mat_type [AIJ形式]	係数行列の格納形式 -mat_type seqaij 逐次版AIJ形式(CSR(CRS)形式) -mat_type mpiaij MPI版AIJ形式(〃) -mat_type mpirowbs MPI版rowbs形式 -mat_type seqdense 逐次版密行列 -mat_type mpidense MPI版密行列 -mat_type seqbaij 逐次版block AIJ形式(〃) -mat_type mpibaij MPI版block AIJ形式(〃)

PETScの魅力の一つとして，makeした実行モジュールに上記のようなオプションを付けることでコマンドライン実行できることが挙げられるが，今回の実習ではバッチ実行のみに制限されているため，コマンドライン記述をスクリプトに書き込む必要がある．

● 上記やその他のオプションの内容の詳細はPETScのuser manual参照．

15. ex1 処理フローの概要

```
inc main(inc argc, char **args ) {
   PetscInitialize
   VecCreate(PETSC_COMM_WORLD,&x);                    解ベクトル x を定義
   VecSetSizes(x,PETSC_DECIDE,n);                     x のサイズを設定
   MatCreate(PETSC_COMM_WORLD,&A);                    行列 A を定義
   MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,n,n);      行列 Aのサイズを設定
   MatSetFromOptions(A);          -mat_typeオプションの反映(デフォルトはAIJ形式)
   MatSetUp(A);          このプログラム内で使えるよう行列 A のデータ構造をセットアップ
 /*    Assemble matrix  */      ###  処理内容は2つ後のスライドを参照  ###
   KSPCreate(PETSC_COMM_WORLD,&ksp);                  使用するKSPを定義
   KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN);
                          行列Aを係数行列として設定し，前処理用行列を定義
   KSPGetPC(ksp,&pc);
   PCSetType(pc,PCJACOBI);                    実際に使用する前処理演算を定義
   KSPSetTolerances(ksp,1.e-5, PETSC_DEFAULT,PETSC_DEFAULT, PETSC_DEFAULT);
   KSPSetFromOptions(ksp);
   KSPSolve(ksp,b,x);                                 求解ルーチン
   KSPView(ksp,PETSC_VIEWER_STDOUT_WORLD);            計算結果の出力
   PetscFinalize();
   return 0;
}
```


15. PETScのex1で使用している係数行列と右辺項

$$Ax = b \qquad b = Ax, \quad x = [1.0, 1.0, \cdots, 1.0]^T$$

$$A = \begin{bmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & 2 \end{bmatrix}$$

係数行列Aの次数n=10(プログラム内で定義)の3重対角行列

15. ex1の係数行列作成の概要

```
PetscInt       n = 10, col[3];
PetscScalar    one = 1.0, value[3];

/*     Assemble matrix  */
 value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
 for (i=1; i<n-1; i++) {
    col[0] = i-1; col[1] = i; col[2] = i+1;
    MatSetValues(A,1,&i,3,col,value,INSERT_VALUES);
 }

 i = n - 1; col[0] = n - 2; col[1] = n - 1;
   MatSetValues(A,1,&i,2,col,value,INSERT_VALUES);

 i = 0; col[0] = 0; col[1] = 1; value[0] = 2.0; value[1] = -1.0;
   MatSetValues(A,1,&i,2,col,value,INSERT_VALUES);
   MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
   MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);

 /*     Set exact solution; then compute right-hand-side vector.  */
 VecSet(u,one);
 MatMult(A,u,b);
```

$b = Au, \quad u = [1.0, 1.0, \cdots, 1.0]^T$

要するに一旦下表を構成して，Aに値を設定している

		value [j] { col [j] }		
		0	1	2
i	0	2.0 {0}	-1.0 {1}	
	1 ～ n-2	-1.0 {i-1}	2.0 {i}	-1.0 {i+1}
	n-1	-1.0 {n-2}	2.0 {n-1}	

※ ex23(MPI並列版)では変数rstart, rendに各ノードのindexを格納して制御.

15. ex1の解のファイル出力例(1/3)

ex1_mod.c ： PETScのex1を伊藤が改造したプログラム（ex1解ベクトル出力版）

```
$ ls
OUT  ex1*  ex1.c  ex11*  ex11.c@  ex11_mod.c  ex11_org.c  ex1_mod.c  ex23*  ex23.c
makefile

$ mv  ex1.c  ex1_org.c
$ mv  ex1    ex1_org

$ ln -s ex1_mod.c ex1.c

$ ls
OUT     ex11*   ex1_mod.c  ex1_org.c  ex23.c
ex1.c@  ex11.c  ex1_org*   ex23*      makefile

$ ls -l ex1.c
lrwxrwxrwx 1 t00003 gt00 9 12月 14 14:13 2012 ex1.c -> ex1_mod.c

$ diff  ex1_org.c  ex1_mod.c
36a37,39
> /* add - begin by itosho */
>   PetscViewer    view_out;
> /* add - end   by itosho */
144a148,154
> /* add - begin by itosho */
> //    VecView(x, PETSC_VIEWER_STDOUT_WORLD);  // To STDOUT file
>     PetscViewerASCIIOpen(PETSC_COMM_WORLD,"Ex1Sol",&view_out);
>     VecView(x, view_out);
>     PetscViewerDestroy(&view_out);
> /* add - end   by itosho */
```

#実習状況により，表示されるファイル群は異なりますが，少なくとも赤文字のファイルは必須.

PETSc も Lis も，このような要領にてサンプルプログラムに手を加え，徐々に自分用のシミュレーションコードにアレンジしていくのが近道（なのだと思う）.

15. ex1の解のファイル出力例(2/3)

```
$ module list
Currently Loaded Modulefiles:
  1) TCSuite/GM-1.2.1-04   2) PETSc/3.3-p4
$ make ex1
  ･･･ 省略 ･･･
$ isub -seq "mpiexec ./ex1"
$ lrt
 ･･･ 省略 ･･･
lrwxrwxrwx 1 t00003 gt00        9 12月 14 14:13 2012 ex1.c -> ex1_mod.c
-rwxr-xr-x 1 t00003 gt00 19289246 12月 14 14:22 2012 ex1*
-rw-r--r-- 1 t00003 gt00       71 12月 14 14:22 2012 Ex1Sol
-rw-r--r-- 1 t00003 gt00      684 12月 14 14:22 2012 OUT

$ cat Ex1Sol
Vector Object:Solution 1 MPI processes
  type: seq
1
1
1
1
1
1
1
1
1
1
1
```

#この例題では，PETSc-complex/3.3-p4であっても実行可能

参考：「15. PETScのex1で使用している係数行列と右辺項」

15. ex1の解のファイル出力例(3/3)

```
$ cat OUT
KSP Object: 1 MPI processes
  type: gmres
    GMRES: restart=30, using Classical (unmodified) Gram-Schmidt Orthogonalization with
no iterative refinement
    GMRES: happy breakdown tolerance 1e-30
  maximum iterations=10000, initial guess is zero
  tolerances:  relative=1e-05, absolute=1e-50, divergence=10000
  left preconditioning
  using PRECONDITIONED norm type for convergence test
PC Object: 1 MPI processes
  type: jacobi
  linear system matrix = precond matrix:
  Matrix Object:   1 MPI processes
    type: seqaij
    rows=10, cols=10
    total: nonzeros=28, allocated nonzeros=50
    total number of mallocs used during MatSetValues calls =0
      not using I-node routines
```

16. KSP法に関する盲点

1. 2つの残差ベクトルについて
   - 求解アルゴリズム中の残差ベクトル
   - 算出された数値解を元の方程式に代入した真の残差ベクトル
2. 残差ベクトルを用いた収束判定について
   - Lis　：　概ね右前処理系に基づく
   - PETSc：デフォルトは左前処理系

参考文献：

[1] 伊藤祥司，杉原正顯，姫野龍太郎，クリロフ部分空間法に対する前処理方式と収束判定について，情報処理学会論文誌コンピューティングシステム(ACS)，Vol.3，No.2，pp.9-19（June 2010）.

[2] 伊藤祥司，杉原正顯，クリロフ部分空間法に対する前処理方向とライブラリ実装における注意点，日本応用数理学会2010年度年会，東京，9月，2010年.

16. 線形方程式（$Ax=b$）求解のための反復解法とその収束性に関する評価の概要

非定常反復法の代表として共役勾配法
(CG: Conjugate Gradient method)

非定常反復法では，残差ベクトルに相当する情報も用いて求解アルゴリズムを構成している（収束判定の計算コスト面で有用である）

適当な初期値 x_0

$$p_0 = r_0 = b - Ax_0$$

収束判定

for $k = 0,1,\cdots$ until $\|r_k\| \le \varepsilon \|b\|$

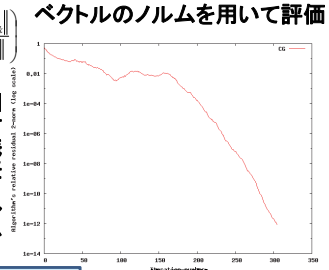
$$\alpha_k = (p_k, r_k)/(p_k, Ap_k)$$
$$x_{k+1} = x_k + \alpha_k p_k$$
$$r_{k+1} = r_k - \alpha_k A p_k$$

← 残差を表すベクトル

$$\beta_k = -(r_{k+1}, Ap_k)/(p_k, Ap_k)$$
$$p_{k+1} = r_{k+1} + \beta_k p_k$$

end

解へ収束する様子を残差ベクトルのノルムを用いて評価



アルゴリズム中の残差ベクトルを評価すれば，
あらためて残差 $r_{k+1} = b - Ax_{k+1}$ を算出せずに済む．

16. main関数での真の残差の算出

当スライドの事柄は，Lis-1.2.95（2012/8/15）版での仕様変更に伴い，（2012/12/20以後の）Lis-1.3.0を用いた講習会では説明不要となった．

test1_org.c test1_mod.c との比較：

```
$ diff test1_org.c test1_mod.c
55a56
>       LIS_REAL                b_nrm2;    // addition
155a157,158
>       lis_vector_nrm2(b, &b_nrm2);    // addition
>
177a181
>               printf("%s: rel_resid = %e¥n¥n",solvername,resid/b_nrm2);  // addition
```

$\|b\|_2$ 用の変数を宣言

$\|b\|_2$ を算出

$\hat{r} = b - A\hat{x}$,　$\hat{x}$：数値解

$\|\hat{r}\|_2 / \|b\|_2$ を算出

```
$ cat OUT
/home/t00003/nfs/lis-1.2.49_seq/test
 ...
BiCG: residual = 6.399933e-15
BiCG: rel_resid = 9.237507e-16
```

$\|\hat{r}\|_2$　Lisのデフォルト (test1_org.c)

$\|\hat{r}\|_2 / \|b\|_2$　当講習会で併用 (test1_mod.c)

16. CGS法を例とした2つの異なる左前処理付きアルゴリズムと収束判定

$Ax = b$　前処理行列：$K \approx A$　残差ベクトル：$r = b - Ax$

伊藤祥司，杉原正顯，姫野龍太郎，情報処理学会論文誌ACS, 3(2), 2010.

左前処理系（Left-preconditioned system）：

$$K^{-1}Ax = K^{-1}b$$

見るからに左前処理付きアルゴリズム

$x_0, r_0 = K^{-1}(b - Ax_0)$,
$\langle r_0^{\#}, r_0 \rangle \neq 0$, e.g., $r_0^{\#} = r_0, \beta_{-1} = 0$,
$k = 0,1,2,\cdots$; Do
　$u_k = r_k + \beta_{k-1} q_{k-1}$,
　$p_k = u_k + \beta_{k-1}(q_{k-1} + \beta_{k-1} p_{k-1})$,
　$\alpha_k = \langle r_0^{\#}, r_k \rangle / \langle r_0^{\#}, K^{-1}Ap_k \rangle$,
　$q_k = u_k - \alpha_k K^{-1}Ap_k$,
　$x_{k+1} = x_k + \alpha_k (u_k + q_k)$,
　$r_{k+1} = r_k - \alpha_k K^{-1}A(u_k + q_k)$,
　$\beta_k = \langle r_0^{\#}, r_{k+1} \rangle / \langle r_0^{\#}, r_k \rangle$,
End Do

右側は両側・左・右3種類の前処理方向の変換で，全て同じアルゴリズムとなる（一致性）．収束判定は下式の通り

収束判定 ○ $\dfrac{\|r_k\|}{\|b\|} = \dfrac{\|b - Ax_k\|}{\|b\|} \le \varepsilon$

残差ベクトル $r_k = b - Ax_k$

残差ベクトル $r_k = K^{-1}(b - Ax_k)$

収束判定

× $\dfrac{\|r_k\|}{\|b\|} = \dfrac{\|K^{-1}(b - Ax_k)\|}{\|b\|} \le \varepsilon$

△ $\dfrac{\|r_k\|}{\|K^{-1}b\|} = \dfrac{\|K^{-1}(b - Ax_k)\|}{\|K^{-1}b\|} \le \varepsilon$

左側は係数行列の部分のみを変換したアルゴリズム．

従来版（Conventional）：

$\tilde{p}_k \Rightarrow K^{-1}p_k, \tilde{u}_k \Rightarrow K^{-1}u_k, \tilde{q}_k \Rightarrow K^{-1}q_k,$
$\tilde{r}_k \Rightarrow K^{-1}r_k, \tilde{r}_0^{\#} \Rightarrow K^{T}r_0^{\#}$

$x_0, r_0 = b - Ax_0$,
$\langle r_0^{\#}, r_0 \rangle \neq 0$, e.g., $r_0^{\#} = r_0, \beta_{-1} = 0$,
$k = 0,1,2,\cdots$; Do
　$u_k = r_k + \beta_{k-1} q_{k-1}$,
　$p_k = u_k + \beta_{k-1}(q_{k-1} + \beta_{k-1} p_{k-1})$,
　$\alpha_k = \langle r_0^{\#}, r_k \rangle / \langle r_0^{\#}, AK^{-1}p_k \rangle$,
　$q_k = u_k - \alpha_k AK^{-1}p_k$,
　$x_{k+1} = x_k + \alpha_k K^{-1}(u_k + q_k)$,
　$r_{k+1} = r_k - \alpha_k AK^{-1}(u_k + q_k)$,
　$\beta_k = \langle r_0^{\#}, r_{k+1} \rangle / \langle r_0^{\#}, r_k \rangle$,
End Do

16. Lisにおける前処理付き アルゴリズムの状況

Lisの非定常反復解法 cg(ID:1)～tfqmr(ID:7), gmres(ID:9)は，右前処理系でありアルゴリズムの収束判定も正常．

ただし，orthomin(k)(ID:8)のみ左前処理系でありながら，別の残差ベクトルを用いて技巧的に収束判定している（好ましくない方法だが，元のSLAPライブラリの設計を踏襲している）．

bicgsafe(ID:13)～minres(ID:22)については，伊藤の方では未確認のため，左前処理系にも関わらず収束判定が
$\|r_k\|/\|b\| \le \varepsilon$ と甘い場合があるかも知れないので注意．

※ 定常反復法（jacobi(ID:10)～sor(ID:12)）と非定常反復法とでは論点が異なるので注意．

16.前処理方向に関する従来からの定義

$Ax = b$　　前処理行列：$K \approx A$　　（残差ベクトル：$r = b - Ax$）

左前処理（l）	両側前処理（b）	右前処理（r）
$K^{-1}A\,x = K^{-1}b$	$K_L^{-1}AK_R^{-1}K_R x = K_L^{-1}b$	$AK^{-1}K\,x = b$
	$K = K_L K_R$	
$K_L = K \quad K_R = I$	K_L：左前処理行列	$K_L = I \quad K_R = K$
	K_R：右前処理行列	I：単位行列

[1] Saad, Y., Iterative Methods for Sparse Linear Systems (2nd ed.), SIAM, 2003.
[2] Van der Vorst, H. A., Iterative Krylov Methosd for Large Linear Systems, Cambridge, 2003.
[3] Barrett, R., et al., Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, (2nd ed.), SIAM, 1994.
[4] 藤野清次，張 紹良，反復法の数理，朝倉書店，1996.

16.クリロフ部分空間(KSP)の基本的な議論

$$A = K - R,\ \ Kx_{k+1} = Rx_k + b,\ \ k = 0,1,2,\cdots$$
$$x_{k+1} = K^{-1}(Rx_k + b) = x_k + K^{-1}(b - Ax_k).$$

前処理変換された残差ベクトル

$r_0 = b - Ax_0$ を用いると，

$$x_k = x_0 + c_0\left(K^{-1}r_0\right) + c_1\left(K^{-1}A\right)\left(K^{-1}r_0\right) + \cdots + c_k\left(K^{-1}A\right)^{k-1}\left(K^{-1}r_0\right)$$
$$\equiv x_0 + z_k, \quad z_k \in \mathcal{K}_k\left(\tilde{A}; \tilde{r}_0\right).$$
$$\mathcal{K}_k\left(\tilde{A}; \tilde{r}_0\right) = \mathrm{Span}\left\{\tilde{r}_0, \tilde{A}\tilde{r}_0, \tilde{A}^2\tilde{r}_0, \cdots, \tilde{A}^{k-1}\tilde{r}_0\right\}$$

KSP法の反復の度に空間が拡張されていく

$\tilde{A}$：前処理変換された係数行列
$\tilde{r}_0$：前処理変換された残差ベクトル

Dongarra, Duff, Sorensen, Van der Vorst, Solving Linear Systems on Vector and Shared Memory Computers, SIAM, 1991.

16.右前処理系の例

$AK^{-1}K\,x = b$

○残差ベクトルは本来の情報を保持している．

$$Kx_{k+1} = Kx_k + (b - Ax_k)$$
$$r_k = b - AK^{-1}Kx_k$$

実質的な残差ベクトル：$r = b - AK^{-1}Kx$

$$\mathcal{K}_k(AK^{-1}; r_0) =$$
$$\mathrm{Span}\left\{r_0, (AK^{-1})r_0, (AK^{-1})^2 r_0, \cdots, (AK^{-1})^{k-1} r_0\right\}$$
$$Kx_k = Kx_0 + \left\{c_0 r_0 + c_1(AK^{-1})r_0 + c_2(AK^{-1})^2 r_0 + \cdots + c_{k-1}(AK^{-1})^{k-1} r_0\right\}$$
$$\Rightarrow Kx_k = Kx_0 + \{c_0 r_0 + c_1(AK^{-1})r_0 + c_2(AK^{-1})^2 r_0 + \cdots + c_{k-1}(AK^{-1})^{k-1} r_0\}$$
$$\in Kx_0 + \mathcal{K}_k\left(AK^{-1}; r_0\right)$$

△生成されるKSPに対し得られる解は，前処理行列が作用したものに基づく．

16.左前処理系の例

$$K^{-1}A\,x = K^{-1}b$$

× 残差ベクトルは本来の情報を保持していない.

$$x_{k+1} = x_k + (K^{-1}b - K^{-1}Ax_k) \qquad r_k = K^{-1}b - K^{-1}Ax_k$$

実質的な残差ベクトル： $r = K^{-1}(b - Ax)$

$$\mathcal{K}_k(K^{-1}A; r_0) =$$

$$\mathrm{Span}\left\{r_0, (K^{-1}A)r_0, (K^{-1}A)^2 r_0, \cdots, (K^{-1}A)^{k-1} r_0\right\}$$

$$x_k = x_0 + \{c_0 r_0 + c_1(K^{-1}A)r_0 + c_2(K^{-1}A)^2 r_0 + \cdots + c_{k-1}(K^{-1}A)^{k-1} r_0\}$$

$$\Rightarrow x_k = x_0 + \{c_0 r_0 + c_1(K^{-1}A)r_0 + c_2(K^{-1}A)^2 r_0 + \cdots + c_{k-1}(K^{-1}A)^{k-1} r_0\}$$

$$\in x_0 + \mathcal{K}_k\left(K^{-1}A; r_0\right)$$

○生成されるKSPに対し得られる解は，前処理行列が作用しないものである.

16.右・左前処理系の例のまとめ

前処理系	ライブラリ	アルゴリズム中の残差ベクトルの情報	生成されるKSPと解の関係
右系	Lisに収められている内の主なKSP法（前半のIDの解法）．その他多くのライブラリやアルゴリズム記述もこちらの系	○保持している $r_k = b - AK^{-1}Kx_k$	△前処理行列が作用したものに基づく $Kx_k \in Kx_0 + \mathcal{K}_k\left(AK^{-1}; r_0\right)$
左系	PETScのデフォルト	×保持していない $r_k = K^{-1}b - K^{-1}Ax_k$	○前処理行列が作用しないもの $x_k \in x_0 + \mathcal{K}_k\left(K^{-1}A; r_0\right)$

実のところ，前処理付きアルゴリズムの構造上，非常に重要なポイントであるにも関わらず，各々の前処理系にてそこそこの効果が得られるため盲点となっており，もっぱら，前処理演算開発の方に注力されてきているのが現状である.

このような解析結果や解決策に関する発表：

[1] 伊藤祥司，杉原正顯，クリロフ部分空間法の前処理系に対する新しい解釈，日本応用数理学会「行列・固有値問題の解法とその応用」研究部会，NII，2009年11月.
[2] 伊藤祥司，杉原正顯，双ランチョス系統の前処理付きアルゴリズムの改善，計算工学講演会論文集 Vol. 15，pp. 171-174，2010年5月.
[3] 伊藤祥司，杉原正顯，様々な前処理付きCGS法に対するクリロフ部分空間に注目した解析，日本応用数理学会 2013年 研究部会 連合発表会，3月14日，東洋大学（白山），2013年.

17. 両ライブラリの習得要領

Lis：

- 係数行列をMatrix Market形式で用意し，test1プログラムを活用（test1自体が単独の数値計算ソフトウェアを構成しているとも考えられる）
- test4，test5の記述を参考にして各自のプログラムを開発する
  test4：係数行列Aの全要素に値を代入する例
  test5：係数行列Aを直接CRS形式で記述する例

PETSc：

- src/ksp/ksp/examples/tutorials{ex1,ex23,ex11}等の記述を参考にして各自のプログラムを開発する（PETScはこの要領で独習するスタイル）
  ex1： 逐次版（MPIコードの1ノード用）：係数行列は実行列
  ex23：MPI並列（ex1の並列版）
  ex11：MPI並列：係数行列は複素行列
- 係数行列のファイルをMatrix Market形式で利用するプログラムは，src/mat/examples/tests/{ex72.c,ex78.c,ex32.c}等を参照して各自で開発する．もしくは，外部ツールの活用もあるが，pestcのバージョンに対応していないと動作しないものも少なく無い.
- PETScの文法に関しては下記ページ参照
  http://www.mcs.anl.gov/petsc/documentation/