Third day [morning session]

Practice to master numerical libraries

RIKEN CCS HPC Summer School 2021
Toshiyuki Imamura, RIKEN CCS
with assistants, Dr. Takeshi Terao and Dr. Takaaki Fukai

Computer simulations
create the future

Regarding software setup on Fugaku

- Set Locale or LANG

```
% export LANG=C
```

- For the third day sessions, you must load appropriate Python modules, such as matplotlib, numpy and scipy, etc. before make and run.

```
% pip3 install --user numpy matplotlib
```

- For additional software on Fugaku login node such as Imagemagick and gnuplot, please activate public and private Spack instance and load several useful OSS packages. See https://www.fugaku.r-ccs.riken.jp/doc_root/ja/user_guides/FugakuSpackGuide-0716-2021/intro.html

When you edit a file,

- **Please carefully check copy&pasted lines.**
- **On ppt or pdf files, some letters are converted in encoding from ASCII to UTF-X.**
  - For example, ‘-’ minus and ‘–’ en-dash (in most cases it is hard to identify on a console terminal).
  - Apostrophe and open-close (single/double) quotation marks.
  - Sometimes backslash ‘＼’ is displayed in the currency mark as in my case ‘¥’. This is because language localization depends your PC environment. Encode is the same as ‘¥’ and ‘＼‘.

  - If you have something wrong in compilation for the copied or pasted lines, please “again” carefully check it or COPY the file by file.

Most of source files and hints are on

/home/ra02006/l00112/school/

PETSc/ → PETSc sample code, a job-script, and Makefile

mpi-petsc_merged_v7.tgz → tarball for CFD+mpi+PETSc

~~**mpi-petsc_merged_v7/** → Karman Vortex CFD codes~~

- **Please 'cd' your work directory, then 'cp -R' the directory ('%' is a prompt, and do not type). You will see**

```
% cd
% cp -R ../l00112/school ./third_day
% cd third_day
% ls
PETSc mpi-petsc_merged_v7.tgz
```

Numerical Library
What is it? For what?

Numerical Library

- **Numerical Library is one of building blocks for ENSURING your advanced programming.**
- **It supports an API for very complex mathematical features, algorithm, schemes, also data handling…**
  - Solving sys. Eqs, FFT, Eigenvalue calculation, SVD, minimization, statistics, etc…
- **There are reference codes.**
  - They might be examples of good (bad) programming.

```
*
*        Form  C := alpha*A*B + beta*C.
*
          DO 90 j = 1,n
             IF (beta.EQ.zero) THEN
                DO 50 i = 1,m
                   c(i,j) = zero
   50           CONTINUE
             ELSE IF (beta.NE.one) THEN
                DO 60 i = 1,m
                   c(i,j) = beta*c(i,j)
   60           CONTINUE
             END IF
             DO 80 l = 1,k
                temp = alpha*b(l,j)
                DO 70 i = 1,m
                   c(i,j) = c(i,j) + temp*a(i,l)
   70           CONTINUE
   80        CONTINUE
   90     CONTINUE
```

http://www.netlib.org/lapack/explore-html/d1/d54/group__double__blas__level3_gaeda3cbd99c8fb834a60a6412878226e1.html

Numerical Library

- **Numerical Library is one of building blocks for ENSURING your advanced programming.**
- **It supports an API for very complex mathematical features, algorithm, schemes, also data handling···**
  - Solving sys. Eqs, FFT, Eigenvalue calculation, SVD, minimization, statistics, etc···
- **There are reference codes.**
  - They might be examples of good (bad) programming.
- **It provides us with better performance and finer accuracy than what you made.**
  - Commercial library: faster and more accurate but expensive
  - Open Source library: fast and free (sometimes faster than commercial library)
  - You must check them before you run your application codes.

Example

- **When you HOPE to SPEED UP your code bottlenecked in matmul, use (or link) an appropriate BLAS (Basic Linear Algebra Subprograms) library!**

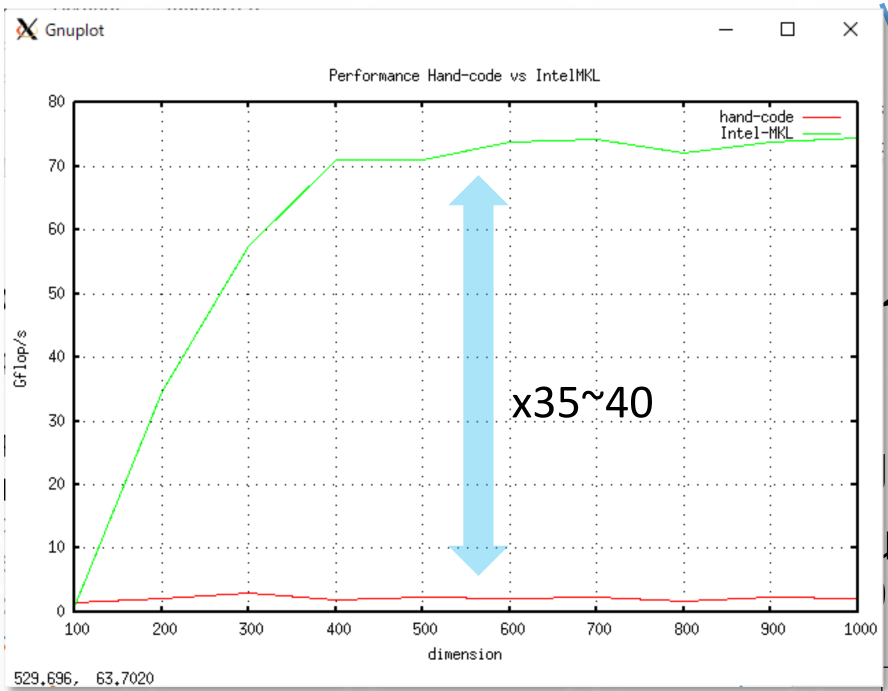
  - Standard API for linear algebra kernels (case of (C)BLAS).
    - GEMM : Matrix-matrix multiplication ($C := \alpha AB + \beta C$)
    - AXPY: linear combination of 2 Vectors ( )
    - NRM2: Norm of a vector, etc. ( )

```
for(i=0; i<N; i++)
 for(j=0; j<N; j++) {
  t = 0.0;
  for(k=0; k<N; k++)
     t += a[i][k]*b[k][j];
  c[i][j] = t;
 }
```

→

```
cblas_dgemm( CblasRowMajor,
             CblasNoTrans, CblasNoTrans,
             N, N, N,
             1.0, a, N, b, N, 0.0, c, N);
```

  - NVIDIA CUBLAS, AMD clMATH, MAGMABLAS(@UTK), KBLAS(@KAUST), ASPEN.K2(@RIKEN) : for GPGPU



UP your code bottlenecked in ppropriate BLAS (Basic Linear rary!

ebra kernels (case of (C)BLAS).

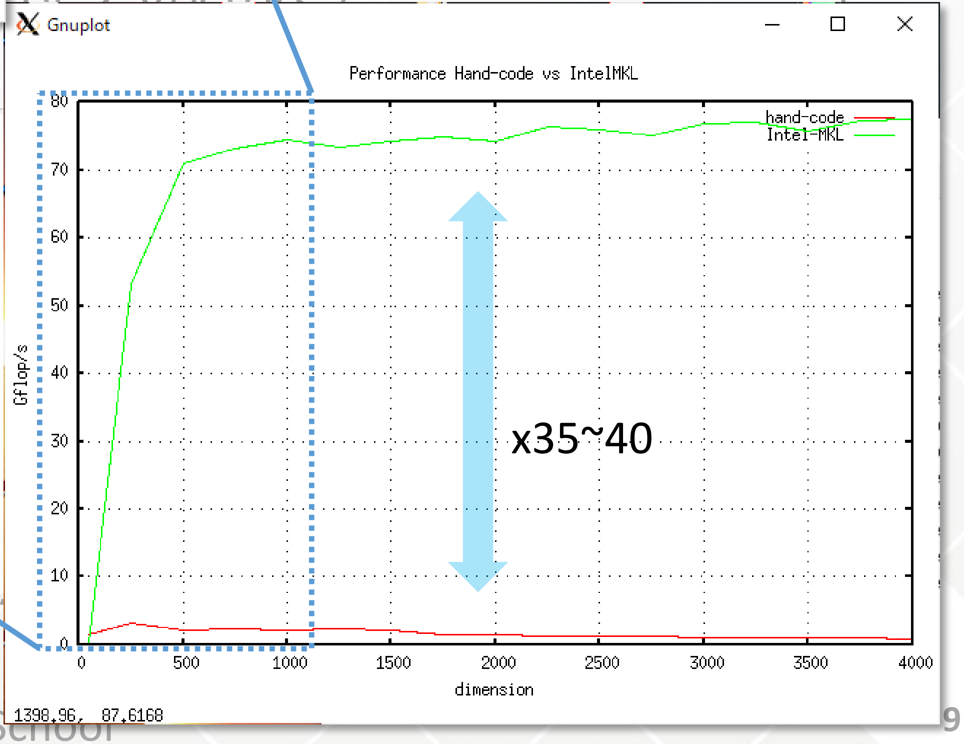
ltiplication

of 2 Vectors (

- NRM2: Norm of a vector,

```
for(i=0; i<N; i++)
 for(j=0; j<N; j++) {
  t = 0.0;
  for(k=0; k<N; k++)
     t += a[i][k]*b[k][j];
  c[i][j] = t;
 }
```

KBLAS(@KAUST), ASPEN.



Example

- **When you HOPE to SPEED UP your code bottlenecked in matmul, use (or link) an appropriate BLAS (Basic Linear Algebra Subprograms) library!**

  - Standard API for linear algebra kernels.
    - GEMM : Matrix-matrix multiplication ($C := \alpha AB + \beta C$)
    - AXPY: linear combination of 2 Vectors ($y := \alpha x + y$)
    - NRM2: Norm of a vector, etc. ($a = \|x\|_2$)

    Reference codes are available from netlib@UTK.

    http://www.netlib.org/BLAS/

  - Commercial: Intel MKL, AMD ACML (free)
  - Open Source: ATLAS(@UTK), GotoBLAS(@TACC), OpenBLAS for general purposed microprocessors
  - nVIDIA CUBLAS, AMD clMATH, MAGMABLAS(@UTK), KBLAS(@KAUST), ASPEN.K2(@RIKEN) : for GPGPU

Other cases

Suggestion: for more complex problems, use followings;

- LAPACK (http://www.netlib.org/lapack/)
- ScaLAPACK (http://www.netlib.org/scalapack/)
- Elemental (http://libelemental.org/)

Dense, General

- EigenExa (http://www.aics.riken.jp/labs/lpnctrt/EigenExa_e.html)
- ELPA (http://elpa.rzg.mpg.de/)

Dense Eigenvalue

- PETSc (http://www.mcs.anl.gov/petsc/)
- Trillions (https://trilinos.org/)

Sparse, General

- ARPACK (http://www.caam.rice.edu/software/ARPACK/)

Sparse, Eigenvalue

- FFTW (http://www.fftw.org/)
- FFTE (http://www.ffte.jp/)
- 2decomp&FFT (http://www.2decomp.org/)

FFT

- MT, MTGP, dSFMT (http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html)

Random number

- GMP big number librart(https://gmplib.org/)
- QD pack, MPACK, and so on

Multi-precision number

Review back to the CFD code

Core computational part in the CFD code

$$\Delta\phi = d$$

$$\frac{\phi^{k}_{i-1,j} - 2\phi^{k+1}_{i,j} + \phi^{k}_{i+1.j}}{\Delta x^2} + \frac{\phi^{k}_{i,j-1} - 2\phi^{k+1}_{i,j} + \phi^{k}_{i.j+1}}{\Delta y^2} - d^{k}_{i,j} = 0$$

$$A\phi = d$$

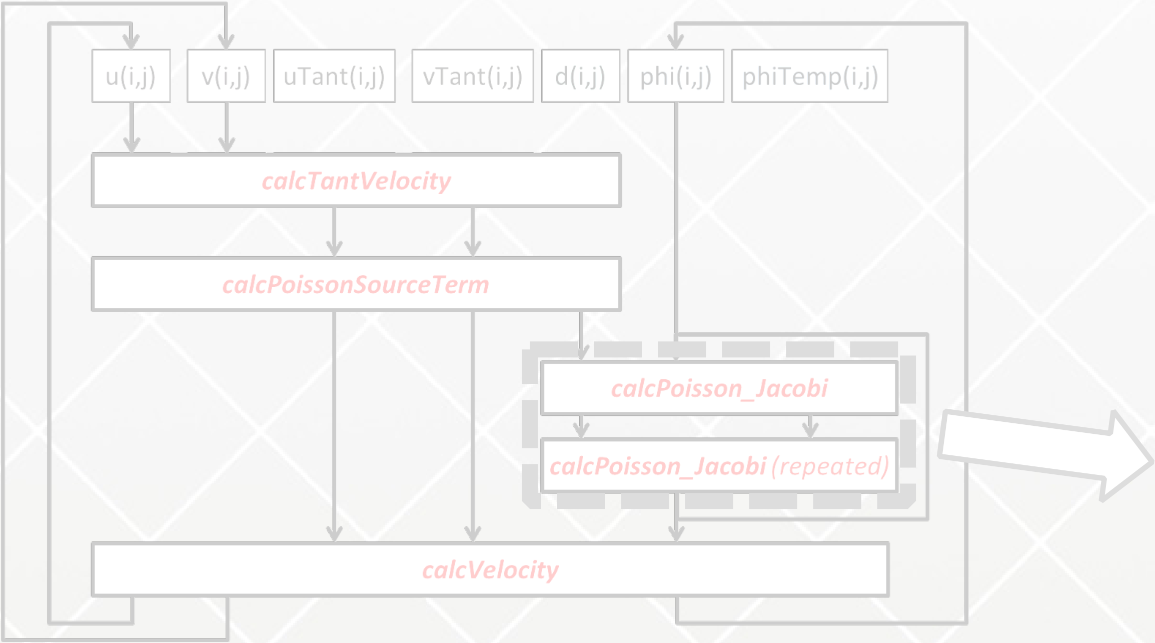
$$A = \frac{1}{\Delta_x^2}(I_{N_y-1} \otimes X_{N_x-1}) + \frac{1}{\Delta_y^2}(X_{N_y-1} \otimes I_{N_x-1})$$

$$X_m = \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & & & & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}_{m \times m}$$

```
for(j=1; j<col_m_1; j++)
    for(i=2; i<row_m_1 - 1; i++)
      *(at(phiT,i,j)) = const1 * ( (L(phi,i+1,j  ) + L(phi,i-1,j  )) * const2 +
                              (L(phi,i  ,j+1) + L(phi,i  ,j-1)) * const3 - L(d,i,j));
```

Solving Poisson' eq

- **You can write a Jacobi iteration code easily. BUT,**
  - wasting your time (programming, execution time).
  - Poisson's eq has a mathematical difficulty of convergence (sometimes diverges).
- **Many algorithms are implemented as library codes.**



You can choose **Faster and reliable** library routine to Solve Poisson'eq.

Today we try to use PETSc libary

Krylov Subspace method
GMRES, CG, GCR, etc..
with Preconditioners

Solving Poisson' eq

- **You can write a Jacobi iteration code easily. BUT,**
  - wasting your time (programming, execution time).
  - Poisson's eq has a mathematical difficulty of convergence (sometimes diverges).
- **Many algorithms are implemented as library codes.**

- **Jacobi iteration**

$$x^{n+1} = D^{-1}((A - D)x^n) - b)$$

Convergence condition is very strict and the Jacobi method sometimes (or often?) diverges or stagnates. (diagonal dominant property is well known as a convergent condition).

Similar algorithms:
Gauss-Seidel, SOR, SSOR, etc.

You can choose **Faster and reliable** library routine to Solve Poisson'eq.

Today we try to use PETSc libary

Krylov Subspace method
GMRES, CG, GCR, etc..
with Preconditioners

Solving Poisson’ eq

- **You can write a Jacobi iteration code easily. BUT,**
  - wasting your time (programming, execution time).
  - Poisson’s eq has a mathematical difficulty of convergence (sometimes diverges).
- **Many algorithms are implemented as library codes.**

- **Jacobi iteration**

$$x^{n+1} = D^{-1}((A - D)x^{n}) - b)$$

Convergence condition is very strict and the Jacobi method sometimes (or often?) diverges or stagnates. (diagonal dominant property is well known as a convergent condition).

Similar algorithms:
Gauss-Seidel, SOR, SSOR, etc.

- **GMRES**

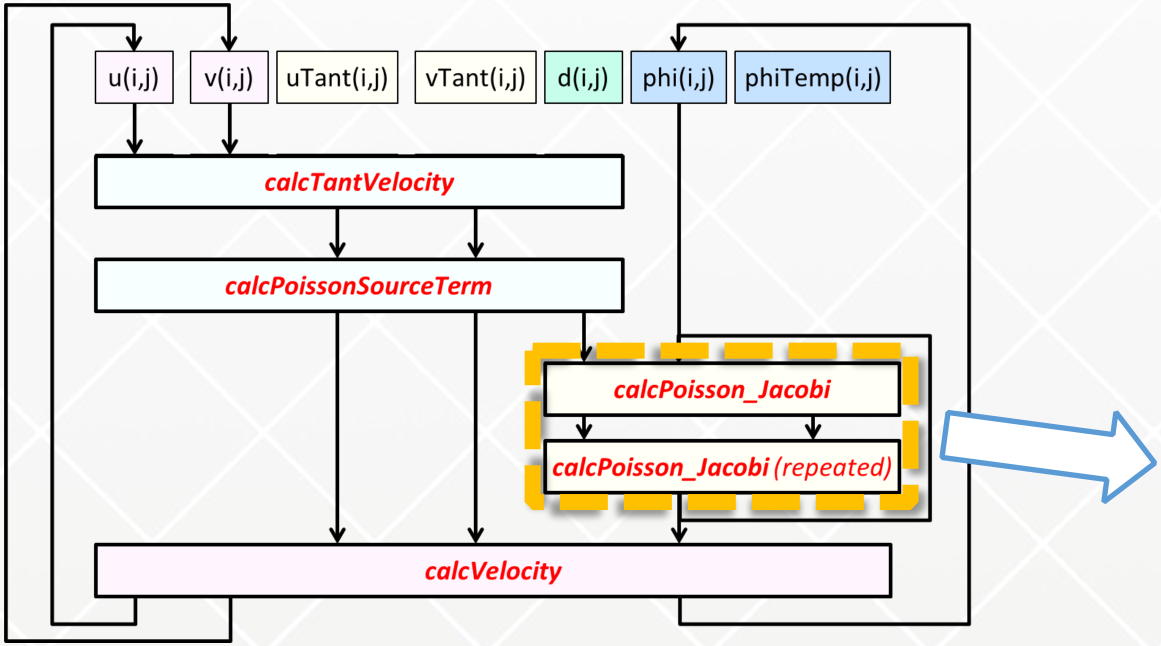
$$\mathrm{argmin}_{y}\{||AVy - b|| | V \in \mathcal{K}^{n}_{A}(v_0)\}$$

Based on Krylov subspace iteration and Arnoldi procedure, it finds a local solution vector by using the Least square approximation within a spanned subspace.

GMRES is known as the best method for a non-symmetric case.

Solving Poisson’ eq

- **You can write a Jacobi iteration code easily. BUT,**
  - wasting your time (programming, execution time).
  - Poisson’s eq has a mathematical difficulty of convergence (sometimes diverges).
- **Many algorithms are implemented as library codes.**



You can choose **Faster and reliable** library routine to Solve Poisson’eq.

Today we try to use PETSc libary

Krylov Subspace method
GMRES, CG, GCR, etc..
with Preconditioners

General use of PETSc(SLEPC)

Official site on PETSc and SLEPC

Hands-on exercises

PETSc/TAO

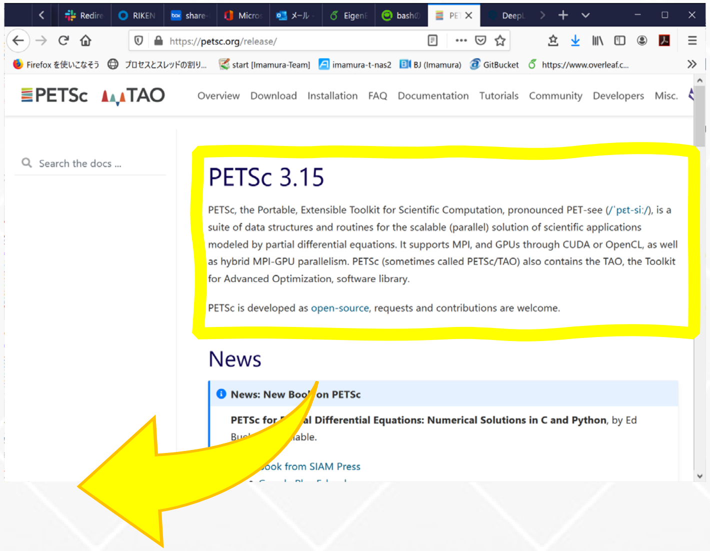
- **Developed by Argonne National Lab. USA**
  - Portable, Extensible Toolkit for Scientific Computation
  - Toolkit for Advanced Optimization

https://petsc.org/

The current version of PETSc is v3.15.3-27-gaef5aac029; released August 24, 2021.

PETSc, the Portable, Extensible Toolkit for Scientific Computation, pronounced PET-see (/ˈpɛt-siː/), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism. PETSc (sometimes called PETSc/TAO) also contains the TAO, the Toolkit for Advanced Optimization, software library.

(cf. PETSc/TAO homepage)



PETSc 3.15

PETSc, the Portable, Extensible Toolkit for Scientific Computation, pronounced PET-see (/ˈpɛt-siː/), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism. PETSc (sometimes called PETSc/TAO) also contains the TAO, the Toolkit for Advanced Optimization, software library.

PETSc is developed as open-source, requests and contributions are welcome.

News

News: New Boo on PETSc

PETSc for al Differential Equations: Numerical Solutions in C and Python, by Ed Bu able.

ook from SIAM Press

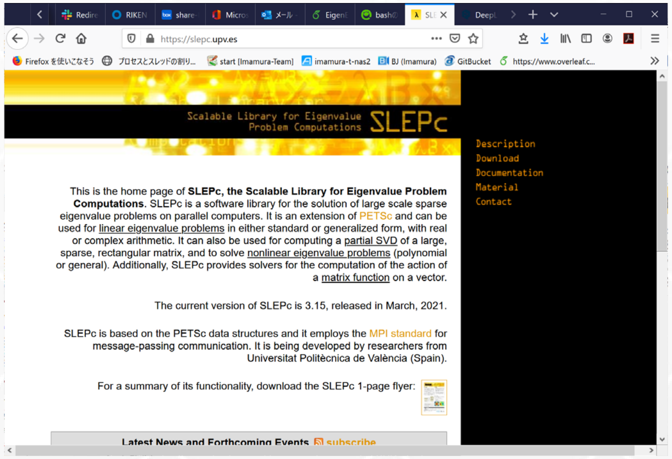
SLEPc

- **Significant plugin modules of PETSc for EVP**
  - developed by Universitat Politecnica de Valencia, Spain

https://slepc.upv.es/

SLEPc is a software library for the solution of large scale sparse eigenvalue problems on parallel computers. It is an extension of PETSc and can be used for linear eigenvalue problems in either standard or generalized form, with real or complex arithmetic. It can also be used for computing a partial SVD of a large, sparse, rectangular matrix, and to solve nonlinear eigenvalue problems (polynomial or general). Additionally, SLEPc provides solvers for the computation of the action of a matrix function on a vector.

The current version of SLEPc is 3.15, released in March, 2021.



This is the home page of **SLEPc, the Scalable Library for Eigenvalue Problem Computations**. SLEPc is a software library for the solution of large scale sparse eigenvalue problems on parallel computers. It is an extension of PETSc and can be used for linear eigenvalue problems in either standard or generalized form, with real or complex arithmetic. It can also be used for computing a partial SVD of a large, sparse, rectangular matrix, and to solve nonlinear eigenvalue problems (polynomial or general). Additionally, SLEPc provides solvers for the computation of the action of a matrix function on a vector.

The current version of SLEPc is 3.15, released in March, 2021.

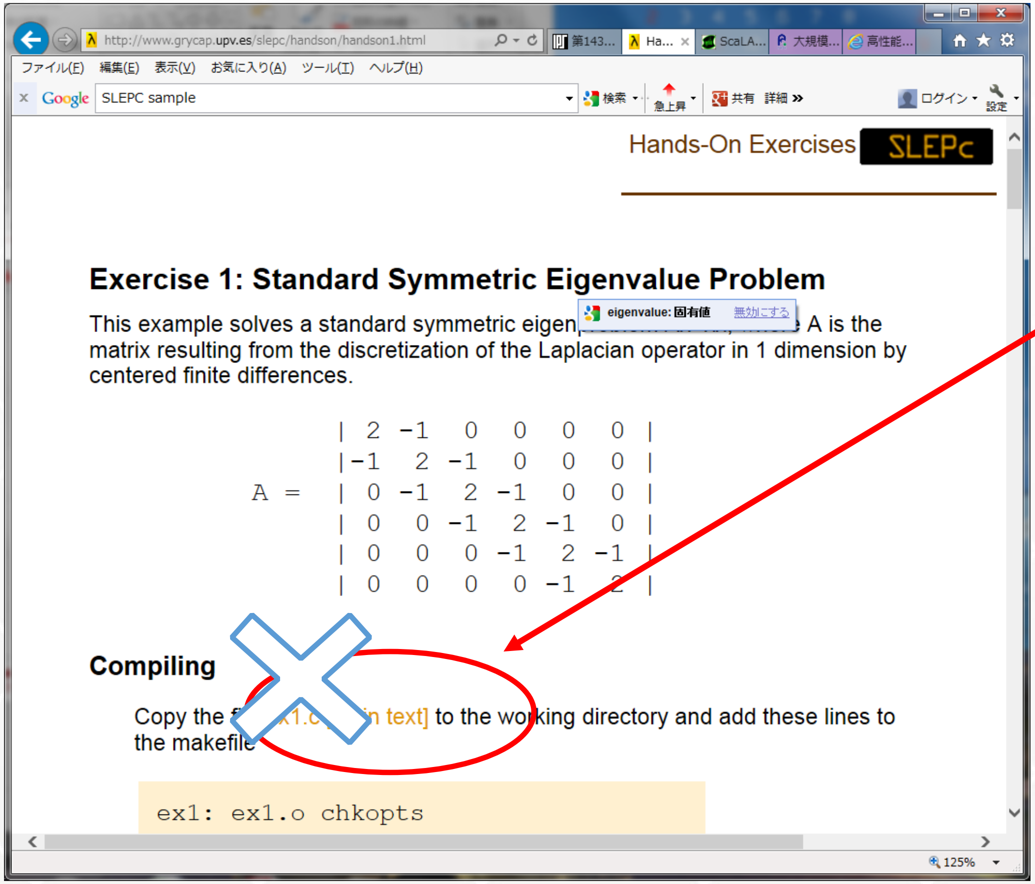
SLEPc is based on the PETSc data structures and it employs the MPI standard for message-passing communication. It is being developed by researchers from Universitat Politècnica de València (Spain).

For a summary of its functionality, download the SLEPc 1-page flyer:

Latest News and Forthcoming Events subscribe

Access to a Sample code

https://slepc.upv.es/handson/handson1.html


http://www.grycap.upv.es/slepc/handson/handson1.html
第143...
Ha...
ScaLA...
大規模...
高性能...
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T) ヘルプ(H)
Google
SLEPC sample
検索
急上昇
共有 詳細 »
ログイン
設定
eigenvalue: 固有値 無効にする
125%

Hands-On Exercises SLEPc

Exercise 1: Standard Symmetric Eigenvalue Problem

This example solves a standard symmetric eigen A is the matrix resulting from the discretization of the Laplacian operator in 1 dimension by centered finite differences.

```
        | 2 -1  0  0  0  0 |
        |-1  2 -1  0  0  0 |
  A  =  | 0 -1  2 -1  0  0 |
        | 0  0 -1  2 -1  0 |
        | 0  0  0 -1  2 -1 |
        | 0  0  0  0 -1  2 |
```

Compiling

Copy the f x1.c n text] to the working directory and add these lines to the makefile

```
ex1: ex1.o chkopts
```

Hyperlinked URL is not available.
Please see ex1.c on the directory ‘PETSc’.

Other tutorial examples are on
/home/ra020006/l00112/school/PETSC/tutorials-petsc and tutorials-slepc

Compile and link

Makefile

```
PETSC_DIR=/home/ra020006/l00112/share/petsc
SLEPC_DIR=/home/ra020006/l00112/share/slepc

CC=mpifccpx -Nclang
FC=mpifrtpx

INCLUDE_DIR=-I${PETSC_DIR}/include -I${SLEPC_DIR}/include
LIBRARY_DIR=-L${PETSC_DIR}/lib -L${SLEPC_DIR}/lib
LIBS=-lslepc -lpetsc -SSL2BLAMP -SCALAPACK -lm

CCFLAGS=-g -Kfast,openmp ${INCLUDE_DIR}
FCFLAGS=-g -Kfast,openmp ${INCLUDE_DIR}
LDFLAGS=-g -Kfast,openmp ${LIBRARY_DIR} ${LIBS}

all: ex1 ex1f

ex1: ex1.o
	${CC} -o $@ $< ${LDFLAGS}
ex1.o: ex1.c
	${CC} -o $@ -c $< ${CCFLAGS}
ex1f: ex1f.o
	${FC} -o $@ $< ${LDFLAGS}
ex1f.o: ex1f.F
	${FC} -o $@ -c $< ${FCFLAGS}

clean:
	¥rm ex1 *.o
```

Use **make** or **make all** command

→→→

ex1 and ex1f are generated.

Job submission

```
#!/bin/bash
#PJM -L "node=1"
#PJM -L "rscgrp=small“  or use “small-torus” if you are going to wait long
#PJM --mpi "max-proc-per-node=4"
#PJM -L "elapse=0:10:00"
#PJM -N "slepc"
#PJM -s

export PETSC_DIR=/home/ra020006/l00112/share/petsc
export SLEPC_DIR=/home/ra020006/l00112/share/slepc
export LD_LIBRARY_PATH=PETSC_DIR/lib:$SLEPC_DIR/lib:$LD_LIBRARY_PATH

export PLE_MPI_STD_EMPTYFILE=off
export OMP_NUM_THREADS=12

mpirun -np 4 ./ex1 -n 540 -m 180 -eps_type gd -eps_nev 4 -eps_monitor -
eps_view
```

```
C version

1-D Laplacian Eigenproblem, n=100

 Number of iterations of the method: 19
 Solution method: krylovschur

 Number of requested eigenvalues: 1
 Stopping condition: tol=1e-08, maxit=100
 Number of converged eigenpairs: 2

           k          ||Ax-kx||/||kx||
   ----------------- ------------------
        3.999033       4.02784e-09
        3.996131       4.31174e-09
```

C version

```
F90 version

1-D Laplacian Eigenproblem, n =100 (Fortran)

 Number of iterations of the method:  19
 Solution method: krylovschur
 Number of requested eigenvalues: 1
 Stopping condition: tol=1.0000E-08, maxit= 100
 Number of converged eigenpairs: 2

          k          ||Ax-kx||/||kx||
  ----------------- ------------------
     3.9990E+00        4.0278E-09
     3.9961E+00        4.3117E-09
```

F90 version

Play with PETSc/SLEPC

- **From the tutorial page,**

```
$ ./ex1 -n 400 -eps_nev 3 -eps_tol 1e-7
```

```
1-D Laplacian Eigenproblem, n=400

 Number of iterations of the method: 100
 Solution method: krylovschur

 Number of requested eigenvalues: 3
 Stopping condition: tol=1e-07, maxit=100
 Number of converged eigenpairs: 1

           k          ||Ax-kx||/||kx||
   ----------------- ------------------
        3.999939       9.48781e-08
```

```
$ ./ex1 -n 400 -eps_nev 3 -eps_ncv 24
```

```
1-D Laplacian Eigenproblem, n=400

 Number of iterations of the method: 60
 Solution method: krylovschur

 Number of requested eigenvalues: 3
 Stopping condition: tol=1e-08, maxit=100
 Number of converged eigenpairs: 5

           k          ||Ax-kx||/||kx||
   ----------------- ------------------
        3.999939       9.48494e-09
        3.999754       7.19493e-09
        3.999448       1.18552e-09
        3.999018       6.43926e-10
        3.998466       1.04213e-09
```

```
$ ./ex1 -n 100 -eps_nev 4 -eps_type lanczos
```

```
1-D Laplacian Eigenproblem, n=100

 Number of iterations of the method: 62
 Solution method: lanczos

 Number of requested eigenvalues: 4
 Stopping condition: tol=1e-08, maxit=100
 Number of converged eigenpairs: 4

           k          ||Ax-kx||/||kx||
   ----------------- ------------------
        3.999033       9.95783e-09
        3.996131       1.97435e-09
        3.991299       9.15231e-09
        3.984540       3.55339e-09
```

Learn the sample code (Pseudo code)

```
SlepcInitialize( PETSC_NULL_CHARACTER, ierr )

MatCreate( PETSC_COMM_WORLD, A, ierr )
MatSetSizes( A, …., n, n, ierr )
MatSetUp( A, ierr )


(…. Calculation of matrix elements and others …..)


ESPCreate( PETSC_COMM_WORLD, eps, ierr )
ESPSetOperators( eps, A, PETSC_NULL_OBJECT, ierr )
EPSSetProblemType( eps, EPS_HEP, ierr )

EPSSolve( eps, ierr )

EPSGetEigenPair( eps, …… )

EPSDestroy( eps, ierr )
SlepcFinalize( ierr )
```

Modern style

1. Initialization
2. Create a Handle for data
3. Setup parameters

4. Create a Handle for solvers
5. Setup parameters

6. Kick the solver

7. Retrieve the results

8. Destroy handles
9. Finalize

How to setup a matrix? (in C)

- **PETSc handles internal data format and interface data flexibly. Because of PETSc management mechanism, user does not to see actual state on memory . Matrix A is dealt with a handler variable, and matrix elements are accessed via a query API.**

```
// Simple matrix format
Mat         A;
EPS         eps;
EPSType     tname;
PetscReal   tol, error, *values;

MatCreate( PETSC_COMM_WORLD, &A )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N )

MatGetOwnershipRange(A, &Istart, &Iend )
MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES )

MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY )
MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY )
```

Create a matrix handler

How to setup a matrix? (in C)

- **PETSc handles internal data format and interface data flexibly. Because of PETSc management mechanism, user does not to see actual state on memory . Matrix A is dealt with a handler variable, and matrix elements are accessed via a query API.**

```
// Simple matrix format
Mat          A;
EPS          eps;
EPSType    tname;
PetscReal  tol, error, *values;

MatCreate( PETSC_COMM_WORLD, &A )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N )

MatGetOwnershipRange(A, &Istart, &Iend )
MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES )

MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY )
MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY )
```

Matrix size MxN

How to setup a matrix? (in C)

- **PETSc handles internal data format and interface data flexibly. Because of PETSc management mechanism, user does not to see actual state on memory . Matrix A is dealt with a handler variable, and matrix elements are accessed via a query API.**

For a mxn block matrix, array values are set.

```
// Simple matrix format
Mat         A;
EPS         eps;
EPSType     tname;
PetscReal   tol, error, *values;

MatCreate( PETSC_COMM_WORLD, &A )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N )

MatGetOwnershipRange(A, &Istart, &Iend )
MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES )

MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY )
MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY )
```

How to setup a matrix? (in C)

- **PETSc handles internal data format and interface data flexibly. Because of PETSc management mechanism, user does not to see actual state on memory . Matrix A is dealt with a handler variable, and matrix elements are accessed via a query API.**

```
// Simple matrix format
Mat         A;
EPS         eps;
EPSType     tname;
PetscReal  tol, error, *values;

MatCreate( PETSC_COMM_WORLD, &A )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE     , N )

MatGetOwnershipRange(A, &Istart, &Iend )
MatSetValues(  A, m, idxm, n, idxn, values  NSERT_VALUES|ADD_VALUES )

MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY )
MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY )
```

Assemble the matrix data set on a distributed manner

How to setup a matrix? (in fortran90)

- **PETSc handles internal data format and interface data flexibly. Because of PETSc management mechanism, user does not to see actual state on memory . Matrix A is dealt with a handler variable, and matrix elements are accessed via a query API.**

```
! Simple matrix format
Mat         A
EPS         eps
EPSType     tname
PetscReal   tol, error, values(:)

call MatCreate( PETSC_COMM_WORLD, A, ierr )
call MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

call MatGetOwnershipRange(A, Istart, Iend, ierr )
call MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES, ierr)

call MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
call MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY, ierr)
```

How to setup a matrix? (in fortran90)

- **PETSc handles internal data format and interface data flexibly. Because of PETSc management mechanism, user does not to see actual state on memory . Matrix A is dealt with a handler variable, and matrix elements are accessed via a query API.**

```
! Simple matrix format
Mat          A
EPS          eps
EPSType    tname
PetscReal  tol, error, values(:)

call MatCreate( PETSC_COMM_WORLD, A, ierr )
call MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

call MatGetOwnershipRange(A, Istart, Iend, ierr )
call MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES, ierr)

call MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
call MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY, ierr)
```

Create a matrix handler

How to setup a matrix? (in fortran90)

- **PETSc handles internal data format and interface data flexibly. Because of PETSc management mechanism, user does not to see actual state on memory . Matrix A is dealt with a handler variable, and matrix elements are accessed via a query API.**

Matrix size MxN

```
! Simple matrix format
Mat         A
EPS         eps
EPSType     tname
PetscReal   tol, error, values(:)

call MatCreate( PETSC_COMM_WORLD, A, ierr )
call MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

call MatGetOwnershipRange(A, Istart, Iend, ierr )
call MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES, ierr)

call MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
call MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY, ierr)
```

How to setup a matrix? (in fortran90)

- **PETSc handles internal data format and interface data flexibly. Because of PETSc management mechanism, user does not to see actual state on memory . Matrix A is dealt with a handler variable, and matrix elements are accessed via a query API.**

For a mxn block matrix, array values are set.

```
! Simple matrix format
Mat          A
EPS          eps
EPSType    tname
PetscReal  tol, error, values(:)

call MatCreate( PETSC_COMM_WORLD, A, ierr )
call MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

call MatGetOwnershipRange(A, Istart, Iend, ierr )
call MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES, ierr)

call MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
call MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY, ierr)
```

How to setup a matrix? (in fortran90)

- **PETSc handles internal data format and interface data flexibly. Because of PETSc management mechanism, user does not to see actual state on memory . Matrix A is dealt with a handler variable, and matrix elements are accessed via a query API.**

```
! Simple matrix format
Mat          A
EPS          eps
EPSType      tname
PetscReal    tol, error, values(:)

call MatCreate( PETSC_COMM_WORLD, A, ierr )
call MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

call MatGetOwnershipRange(A, Istart, Iend, ierr )
call MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES, ierr)

call MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
call MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY, ierr)
```

Assemble the matrix data set on a distributed manner

Play back the CFD code and link it with PETSc

Core computational part in the CFD code

- **Solving a Poisson equation**

$$\Delta\phi = d$$

- **Discretization:**

$$\frac{\phi^{k}_{i-1,j} - 2\phi^{k+1}_{i,j} + \phi^{k}_{i+1.j}}{\Delta x^2} + \frac{\phi^{k}_{i,j-1} - 2\phi^{k+1}_{i,j} + \phi^{k}_{i.j+1}}{\Delta y^2} - d^{k}_{i,j} = 0$$

$$A\phi = d$$

$$A = \frac{1}{\Delta_x^2}(I_{N_y-1} \otimes X_{N_x-1}) + \frac{1}{\Delta_y^2}(X_{N_y-1} \otimes I_{N_x-1})$$

$$X_m = \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & & & & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}_{m\times m}$$

- **Finally, Core iteration: it is a well-know iteration**

```
for(j=1; j<col_m_1; j++)
    for(i=2; i<row_m_1 - 1; i++)
      *(at(phiT,i,j)) = const1 * ( (L(phi,i+1,j  ) + L(phi,i-1,j  )) * const2 +
                                    (L(phi,i  ,j+1) + L(phi,i  ,j-1)) * const3 - L(d,i,j));
```

Use a PETSc KSP solver

- **What I did were,**
  - Download a sample source code
  - Modify main.cpp and cfd.cpp
  - Key points are i) initialization of PETSc environment and matrix generation, ii) calling a solver routine.

PETSc supports DMDA (distributed multi-dimentional array format) for vector data and a stencil mechanism for matrix representation.

- I Prepared 3 call-back functions, and 1 utility function;
  - **extern PetscErrorCode ComputeStencil (KSP, Mat, Mat, void *);**
  - **extern PetscErrorCode ComputeRHS (KSP, Vec, void *);**
  - **extern PetscErrorCode SetInitialGuess (KSP, Vec, void *);**
  - **extern PetscErrorCode GetComputedResults (KSP, void *);**
- Then, associated them with a DM handler.
- KSPSolve() organized RHS and MatVec, etc.

Use a PETSc KSP solver

- **What I did were,**
  - Download a sample source code
  - Modify main.cpp and cfd.cpp
  - Key points are i) initialization of PETSc environment and matrix generation, ii) calling a solver routine.

PETSc supports DMDA (distributed multi-dimentional array format) for vector data and a stencil mechanism for matrix representation.

- I Prepared 3 call-back functions, and 1 utility function;
  - **extern PetscErrorCode ComputeStencil (KSP, Mat, Mat, void *);**
  - **extern PetscErrorCode ComputeRHS (KSP, Vec, void *);**
  - **extern PetscErrorCode SetInitialGuess (KSP, Vec, void *);**
  - **extern PetscErrorCode GetComputedResults (KSP, void *);**
- Then, associated them with a DM handler.
- KSPSolve() organized RHS and MatVec, etc.

Use a PETSc KSP solver

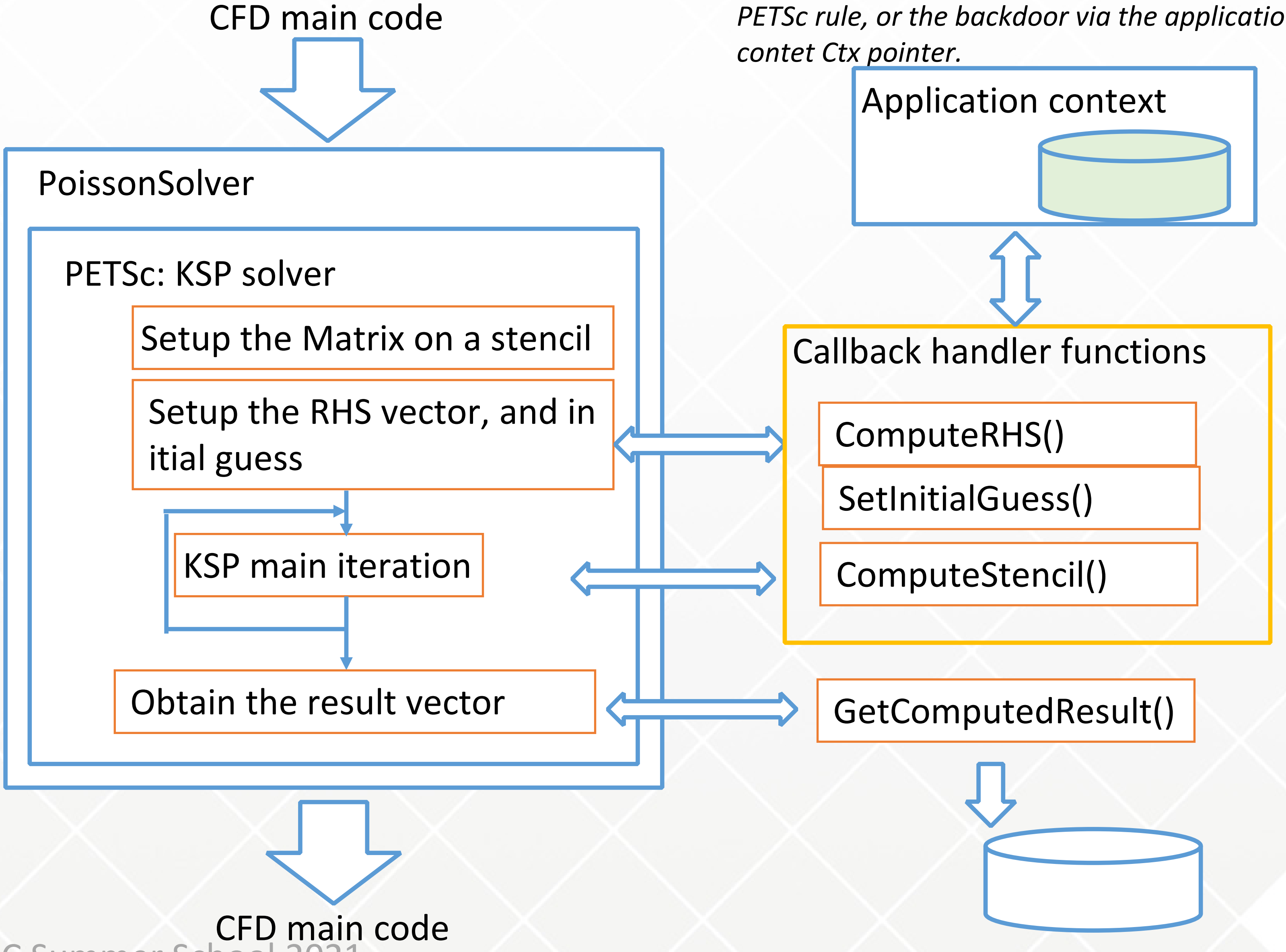
- **What I did were,**
  - Download a sample source code
  - Modify main.cpp and cfd.cpp
  - Key points are i) initialization of PETSc environment and matrix generation, ii) calling a solver routine.

PETSc supports DMDA (distributed multi-dimentional array format) for vector data and a stencil mechanism for matrix representation.

- I Prepared 3 call-back functions, and 1 utility function;
  - **extern PetscErrorCode ComputeStencil (KSP, Mat, Mat, void *);**
  - **extern PetscErrorCode ComputeRHS (KSP, Vec, void *);**
  - **extern PetscErrorCode SetInitialGuess (KSP, Vec, void *);**
  - **extern PetscErrorCode GetComputedResults (KSP, void *);**
- Then, associated them with a DM handler.
- KSPSolve() organized RHS and MatVec, etc.

(PETSc) KSP solver and CFD code

Once KSP starts up, user can access the data within PETSc rule, or the backdoor via the application contet Ctx pointer.



Use a PETSc KSP solver

1. **ComputeStencil**
   - Jacobi iteration is written in a 5-point Stencil fashion. Any stencil codes are

$$u^{new}_{i,j} := au_{i-1,j} + bu_{i,j-1} + cu_{i,j} + du_{i+1,j} + eu_{i,j+1} + f_{i,j}$$

   - Translate above relation into a Matrix-vector product.

2. **ComputeRHS**
   - Update the vector which appears in Right hand side

3. **SetInitialGuess**
   - Set up an initial guess for Krylov iterative solver

4. **GetComputedResult.**
   - Retrieve the result computed by using PETS.

Use a PETSc KSP solver

1. **ComputeStencil**
   - Jacobi iteration is written in a 5-point Stencil fashion. Any stencil codes are

$$u_{i,j}^{new} := au_{i-1,j} + bu_{i,j-1} + cu_{i,j} + du_{i+1,j} + eu_{i,j+1} + f_{i,j}$$

*** Key functions ***

1. MatSetStencil(Mat m, int dim, int dims[], int starts[], int dof)
It sets the grid information for setting values into a matrix

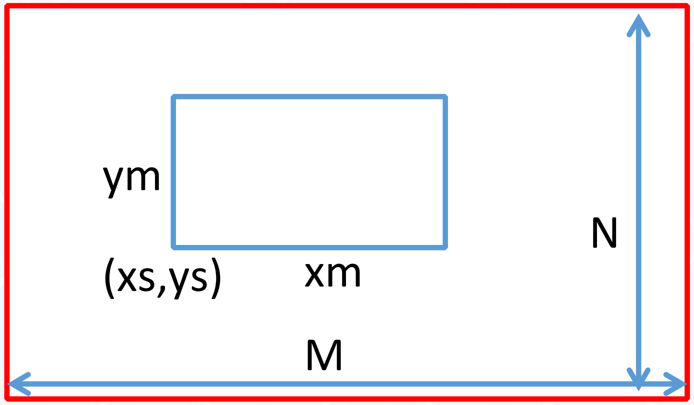
2. KSPSetComputeOperators(KSP ksp, (*func)(), void *ctx)
It set a Stencil callback routine to contexts such as DM/user-defined handler

4. **GetComputedResult.**
   - Retrieve the result computed by using PETS.

ComputeStencil

```
ComputeStencil (KSP ksp, Mat A, Mat Aac, void *ctx)
{
  DM da;
  int i, j, M, N, xm, ym, xs, ys;
  grid2D *g = (grid2D *) ctx;

  PetscFunctionBeginUser;

  KSPGetDM (ksp, &da);
  DMDAGetInfo (da, 0, &M, &N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
  DMDAGetCorners (da, &xs, &ys, 0, &xm, &ym, 0);

  const PetscScalar const2 = 1.0 / DX2;
  const PetscScalar const3 = 1.0 / DY2;
  const PetscScalar inv_const1 = 2 * (const2 + const3);

  for (j = ys; j < ys + ym; j++) {
    for (i = xs; i < xs + xm; i++) {
      PetscScalar v[5];
      MatStencil global_index, local_indices[5];

      global_index.i = i;
      global_index.j = j;
```

- Obtain DA and DMDA infos
- Get the global domain info, corresponding to the local process

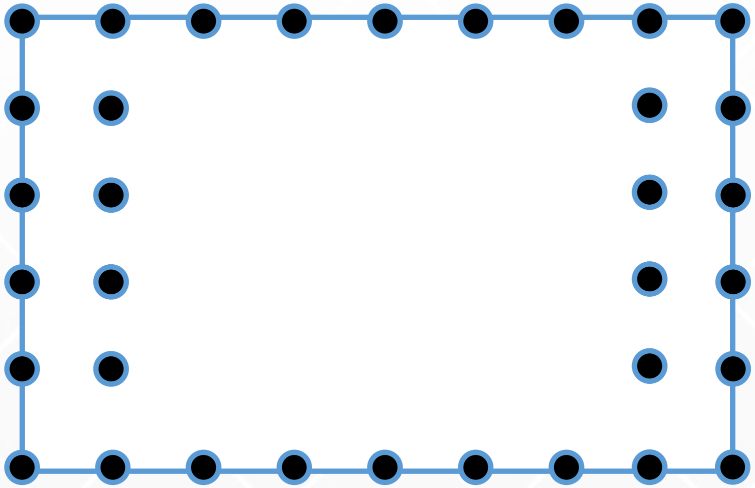


- Declaration of a work-array for the 5-point stencil

ComputeStencil

```
if (i <= 1 || j <= 0 || i >= M - 2 || j >= N - 1) {

/*  Regular Boundary : not changed  */

      v[0] = 1;
      local_indices[0].i = i;
      local_indices[0].j = j;
      MatSetValuesStencil (Aac, 1, &global_index, 1,
                           local_indices, v, INSERT_VALUES);
    }
    else {

      int is_object = 0;

      int ctr_obj;
      for( ctr_obj=0; ctr_obj<g->objs.n; ctr_obj++ ) {
/*  Boundary of the object */

      const int obj_x = (int) g->objs.obj[ctr_obj].x;
      const int obj_y = (int) g->objs.obj[ctr_obj].y;
      const int obj_w = (int) g->objs.obj[ctr_obj].w;
      const int obj_h = (int) g->objs.obj[ctr_obj].h;
```

Boundary grid points

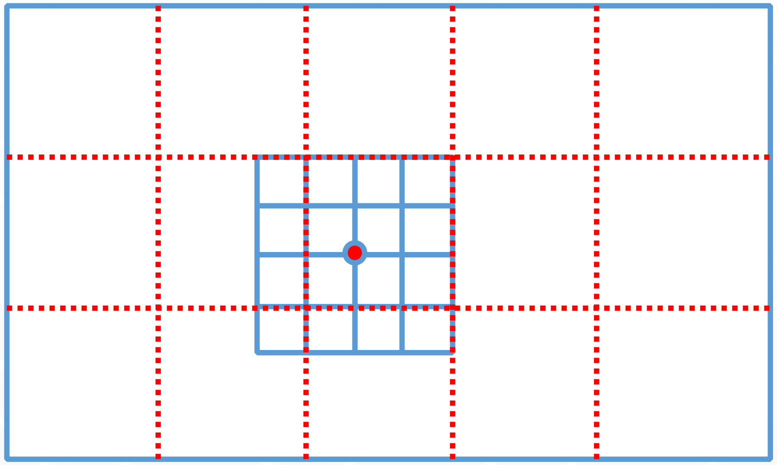


For the inlet/outlet flow, both left/right wall is boubled (i.e.i=0,1 and M-2,M-1)

- Compute the object region in global grid.
- (obj_x, obj_y) : Center point
- (obj_w, obj_h): Shape, width/height

ComputeStencil

```
     const int sta_i = (int) (obj_x - obj_w / 2);
     const int end_i = (int) (sta_i + obj_w);
     const int sta_j = (int) (obj_y - obj_h / 2);
     const int end_j = (int) (sta_j + obj_h);

 if (i >= sta_i && i <= end_i && j >= sta_j && j <= end_j)
{
      is_object = 1;

      int num, numi, numj;
      num = numi = numj = 0;

      if (i == sta_i) {
        v[num] = -DY;
        local_indices[num].i = i - 1;
        local_indices[num].j = j;
        num++;
        numi++;
      }
      else if (i == end_i) {
        v[num] = -DY;
        local_indices[num].i = i + 1;
```



- As you can see the above figure, the relation between local grid and global one differs.
- Count up the touched boundary
- and store the coefficients on the 5-point stencil

ComputeStencil

```
  local_indices[num].j = j;
  num++;
  numi++;
}
else {
if (j == sta_j) {
  v[num] = -DX;
  local_indices[num].i = i;
  local_indices[num].j = j - 1;
  num++;
  numj++;
}
else if (j == end_j) {
  v[num] = -DX;
  local_indices[num].i = i;
  local_indices[num].j = j + 1;
  num++;
  numj++;
}
}
```

- Check whether on the object wall, if so, setup special boundary condition according to the numerical scheme.

ComputeStencil

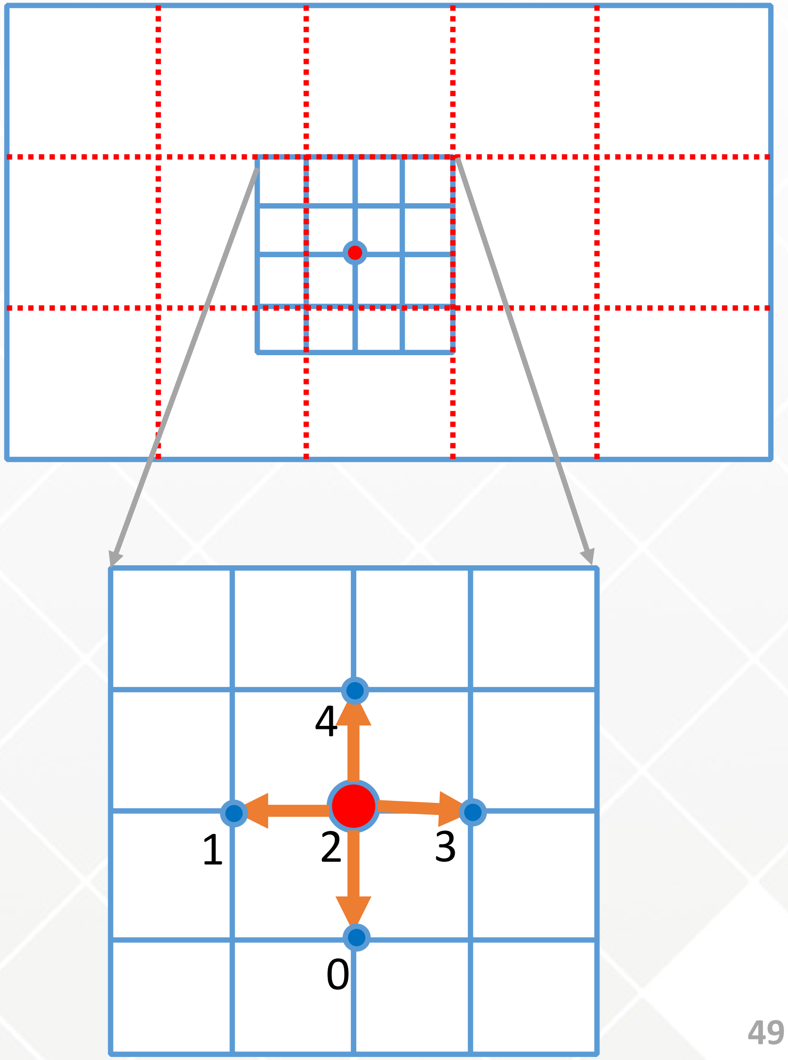
```
  if (num > 0) {        // surface of the object
    v[num] = (numj * DX + numi * DY);
  }
  else {                // object internal
    v[0] = 1;
  }
  local_indices[num].i = i;
  local_indices[num].j = j;
  num++;
  MatSetValuesStencil (Aac, 1,
        &global_index,
        num, local_indices, v, INSERT_VALUES);
}

}
```

- Set up the coefficient of the central point in the 5-point stencil.
- If the point is object internal, output is invariant to the input.

ComputeStencil

```
if ( ! is_object ) {

  v[0] = const3;
  local_indices[0].i = i;
  local_indices[0].j = j - 1;
  v[1] = const2;
  local_indices[1].i = i - 1;
  local_indices[1].j = j;
  v[2] = -inv_const1;
  local_indices[2].i = i;
  local_indices[2].j = j;
  v[3] = const2;
  local_indices[3].i = i + 1;
  local_indices[3].j = j;
  v[4] = const3;
  local_indices[4].i = i;
  local_indices[4].j = j + 1;
  MatSetValuesStencil (Aac, 1,
                     &global_index, 5,
                     local_indices, v, INSERT_VALUES);
```

- Set up the coefficient of the central point in the 5-point stencil on the Fluid region.



ComputeStencil

```
    }
   }
  }
 }
 MatAssemblyBegin (Aac, MAT_FINAL_ASSEMBLY);
 MatAssemblyEnd (Aac, MAT_FINAL_ASSEMBLY);
 PetscFunctionReturn (0);
}
```

- Do not forget to finalize the assembling the matrix data elements.

Use a PETSc KSP solver

1. ComputeStencil

* Key functions *

1. DMDAVecGetArray(DM da, Vec vec, void *array)
   It returns a pointer (an underlying vector) with global-index

2. KSPSetComputeRHS (KSP ksp, (*func)(), void *ctx)
   It sets an RHS callback routine and a user-defined context

2. **ComputeRHS**
   - Update the vector which appears in Right hand side

3. **SetInitialGuess**
   - Set up an initial guess for Krylov iterative solver

4. **GetComputedResult.**
   - Retrieve the result computed by using PETS.

ComputeRHS

```
ComputeRHS (KSP ksp, Vec b, void *ctx)
{
  DM da;
  int i, j,M, N, xm, ym, xs, ys;
  grid2D *g = (grid2D *) ctx;

  PetscFunctionBeginUser;

  array2D *phi = &(g->phi);
  array2D *d = &(g->d);
  array2D *u = &(g->u);
  array2D *v = &(g->v);

  KSPGetDM (ksp, &da);
  DMDAGetInfo (da, 0, &M, &N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
  DMDAGetCorners (da, &xs, &ys, 0, &xm, &ym, 0);
  PetscScalar **array;
  DMDAVecGetArray (da, b, &array);
```

- Ctx is a user-defined context handle, which contains any info to calculate matrix and vectors.
- Here, the pointer to the grid2D structured data set was assigned.

- DMDAVecGetArray() returns a multiple dimension array that shares data with the underlying vector and is indexed using the global dimensions. (see. PETSc DMDA manual)

ComputeRHS

```
#pragma omp parallel for private(i)
 for (j = ys; j < ys + ym; j++) {
  for (i = xs; i < xs + xm; i++) {

   double tmp = 0.0;
   if (i <= 1 || j <= 1 || i >= M - 2 || j >= N - 2) {
/*  Boundary wall 2-point layer */
    tmp = L (phi, i-xs, j-ys);
   }
   else {

    int is_object = 0;

    int ctr_obj;
    for( ctr_obj=0; ctr_obj<g->objs.n; ctr_obj++ ) {

    const int obj_x = (int) g->objs.obj[ctr_obj].x;
    const int obj_y = (int) g->objs.obj[ctr_obj].y;
    const int obj_w = (int) g->objs.obj[ctr_obj].w;
    const int obj_h = (int) g->objs.obj[ctr_obj].h;
```

- Set the ctr_obj-th object information

ComputeRHS

```
    const int sta_i = (int) (obj_x - obj_w / 2);
    const int end_i = (int) (sta_i + obj_w);
    const int sta_j = (int) (obj_y - obj_h / 2);
    const int end_j = (int) (sta_j + obj_h);

// Boundary of the object
    if (i >= sta_i && i <= end_i && j >= sta_j && j <=
end_j) {
      is_object = 1;

// in case surface of the object
      int  num = 0;
      if ( i == sta_i ) { num++;
        tmp += DY * (2 * NU / DX) * L (u, i-xs - 1, j-ys);
      }
      else if ( i == end_i ) { num++;
        tmp += DY * (2 * NU / DX) * L (u, i-xs + 1, j-ys);
      }
      else {
        if ( j == sta_j ) { num++;
          tmp += DX * (2 * NU / DY) * L (v, i-xs, j-ys - 1);
        }
```

- Check Region of the object
- Check Boundary of the object

ComputeRHS

```
     else {
      if ( j == sta_j ) { num++;
       tmp += DX * (2 * NU / DY) * L (v, i-xs, j-ys - 1);
      }
      else if ( j == end_j ) { num++;
       tmp += DX * (2 * NU / DY) * L (v, i-xs, j-ys + 1);
      }
    }
// in case inside of the object
     if ( num == 0 ) { tmp = -150.; }
    }
    }
    if ( ! is_object ) {
     tmp = L (d, i-xs, j-ys);
    }
   }
   array[j][i] = tmp;
  }
 }
```

- Object in the Fluid domain
- According to the boundary condition, RHS is computed.
- -150 is artificially set as the smallest number (pressure)

ComputeRHS

```
DMDAVecRestoreArray (da, b, &array);
  VecAssemblyBegin (b);
  VecAssemblyEnd (b);

 PetscFunctionReturn (0);
}
```

- Regular points in the Fluid domain
- Restores a multiple dimension array obtained with DMDAVecGetArray() .
- Do not forget to finalize the assembling the vector data elements.

Use a PETSc KSP solver

1. **ComputeStencil**
   - Jacobi iteration is written in a 5-point Stencil fashion. Any stencil codes are

$$u_{i,j}^{\mathrm{new}} := au_{i-1,j} + bu_{i,j-1} + cu_{i,j} + du_{i+1,j} + eu_{i,j+1} + f_{i,j}$$

   - Translate above relation into a Matrix-vector product.

2. **ComputeRHS**
   - Update the vector which appears in Right hand side

3. **SetInitialGuess**
   - Set up an initial guess for Krylov iterative solver

4. **GetComputedResult.**
   - Retrieve the result computed by using PETS.

SetInitialGuess

```
SetInitialGuess (KSP ksp, Vec b, void *ctx)
{
  DM da;
  int i, j, xm, ym, xs, ys;
  grid2D *g = (grid2D *) ctx;
  PetscFunctionBeginUser;
  KSPGetDM (ksp, &da);
  DMDAGetCorners (da, &xs, &ys, 0, &xm, &ym, 0);
  double **array;
  DMDAVecGetArray (da, b, &array);
  array2D *phi = &(g->phi);
#pragma omp parallel for private(i)
  for (j = ys; j < ys+ym; j++) {
    for (i = xs; i < xs+xm; i++) {
      array[j][i] = (L (phi, i-xs-1, j-ys) + L (phi, i-xs, j-ys-1) +
                     L (phi, i-xs+1, j-ys) + L (phi, i-xs, j-ys+1))/4;
    }
  }
  DMDAVecRestoreArray (da, b, &array);
  VecAssemblyBegin (b);
  VecAssemblyEnd (b);
  PetscFunctionReturn (0);
}
```

- For the initial guess for KSP, just use a mean value of four neighboring points. (similar to the Jacobi solver)

- Do not forget to finalize the assembling the vector data elements.

Use a PETSc KSP solver

1. **ComputeStencil**
   - Jacobi iteration is written in a 5-point Stencil fashion. Any stencil codes are

$$u^{new}_{i,j} := au_{i-1,j} + bu_{i,j-1} + cu_{i,j} + du_{i+1,j} + eu_{i,j+1} + f_{i,j}$$

   - Translate above relation into a Matrix-vector product.

2. **ComputeRHS**
   - Update the vector which appears in Right hand side

3. **SetInitialGuess**
   - Set up an initial guess for Krylov iterative solver

4. **GetComputedResult.**
   - Retrieve the result computed by using PETS.

GetComputedResult

```
GetComputedResults (KSP ksp, void *ctx)
{
  DM da;
  int i, j, xm, ym, xs, ys;
  grid2D *g = (grid2D *) ctx;
  PetscFunctionBeginUser;

  KSPGetDM (ksp, &da);
  DMDAGetCorners (da, &xs, &ys, 0, &xm, &ym, 0);
  Vec b;
  KSPGetSolution (ksp, &b);
  double **array;
  DMDAVecGetArray (da, b, &array);
  array2D *phi = &(g->phi);
#pragma omp parallel for private(i)
  for (j = ys; j < ys+ym; j++) {
    for (i = xs; i < xs+xm; i++) {
      *(at (phi, i-xs, j-ys)) = array[j][i];
    }
  }
  PetscFunctionReturn (0);
}
```

- Obtain the corresponding data, while DMDA holds Halo (overlap region).
- Also, needs to reshape the data structure.

Use a PETSc KSP solver

1. **ComputeStencil**
   - Jacobi iteration is written in a 5-point Stencil fashion. Any stencil codes are

$$u^{\mathrm{new}}_{i,j} := au_{i-1,j} + bu_{i,j-1} + cu_{i,j} + du_{i+1,j} + eu_{i,j+1} + f_{i,j}$$

   - Translate above relation into a Matrix-vector product.

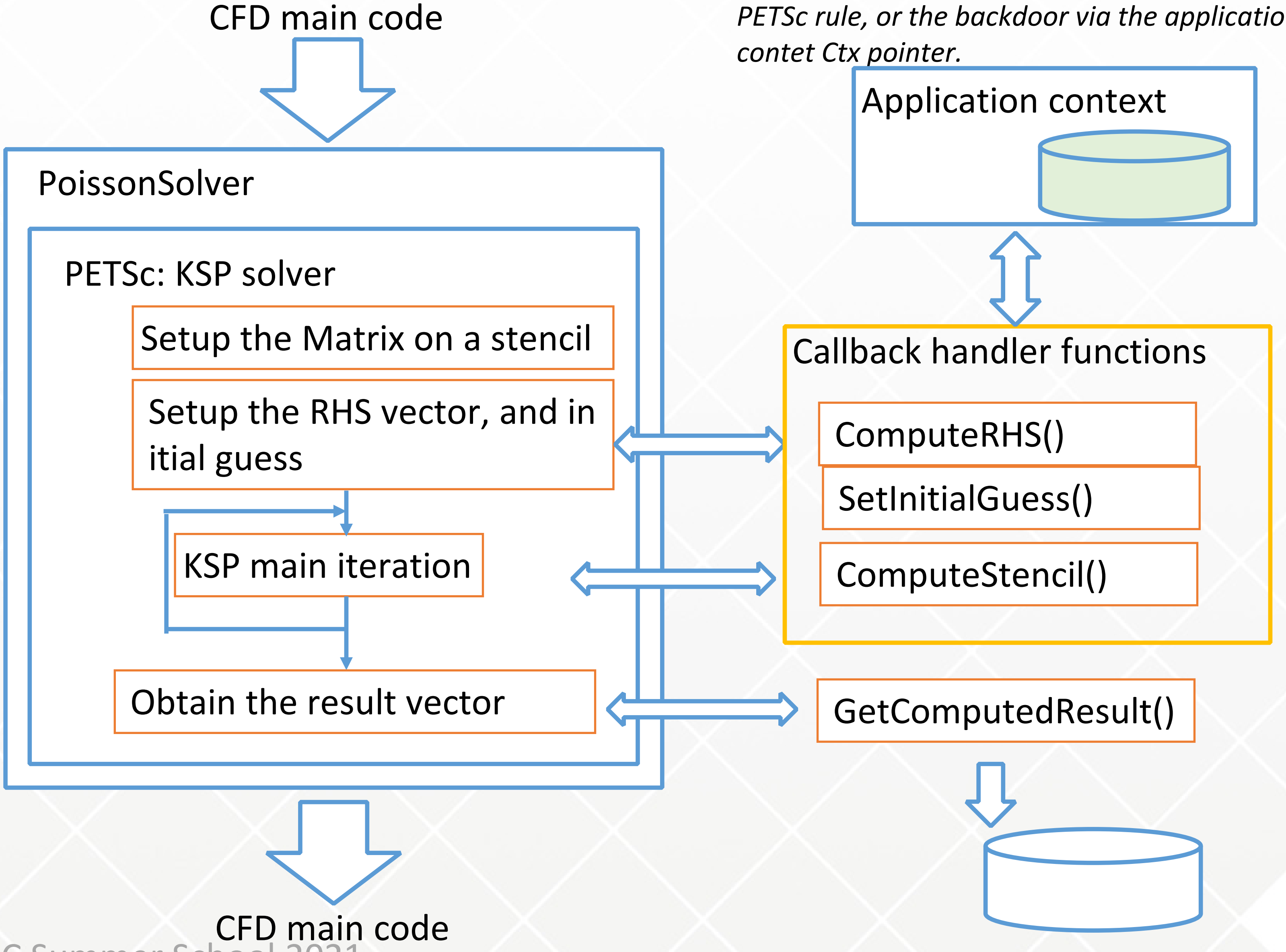
2. **ComputeRHS**
   - Update the vector which appears in Right hand side

3. **SetInitialGuess**
   - Set up an initial guess for Krylov iterative solver

4. **GetComputedResult.**
   - Retrieve the result computed by using PETS.

(PETSc) KSP solver and CFD code

Once KSP starts up, user can access the data within PETSc rule, or the backdoor via the application contet Ctx pointer.



Integrated or registration part for the call-back functions

```
KSP  ksp;  // global variable, sorry it is not a good manner
.
void grid2D_initialize( … )
{
   …
  KSPCreate (PETSC_COMM_WORLD, &ksp);
  DM da;
  DMDACreate2d (PETSC_COMM_WORLD,
                 DM_BOUNDARY_NONE, DM_BOUNDARY_NONE,
                DMDA_STENCIL_STAR, row, col, Px, Py, 1, 1,
                 NULL, NULL, &da);
  DMSetUp (da);
  KSPSetDM (ksp, (DM) da);
  DMSetApplicationContext (da, (void *) g);
  KSPSetComputeInitialGuess (ksp, SetInitialGuess, (void *) g);
  KSPSetComputeRHS (ksp, ComputeRHS, (void *) g);
  KSPSetComputeOperators (ksp, ComputeStencil, (void *) g);

  …
}
```

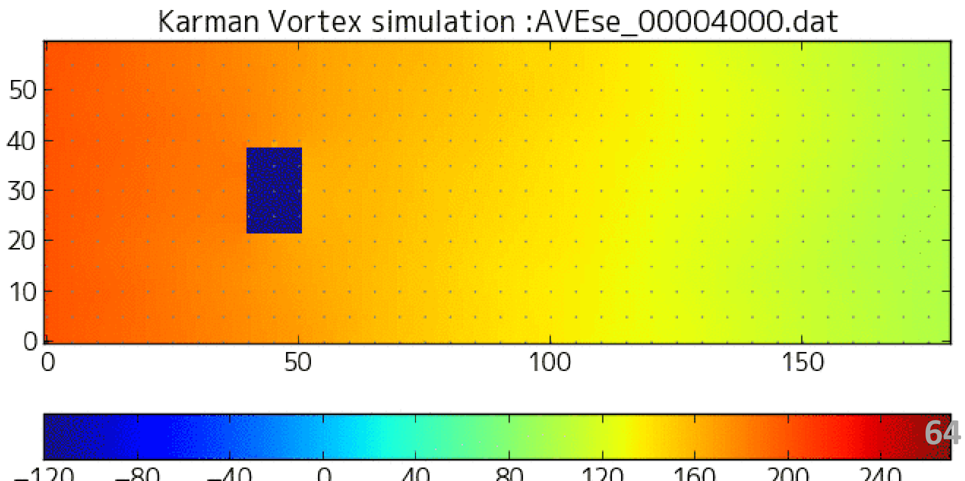
- These are super-important functions!

Let’s Run the CFD code on OBCX

Copy mpi-petsc_merged_v7.tgz from /home/ra020006/l00112/school

```
% tar zxvf mpi-petsc_merged_v7.tgz
% cd mpi-petsc_merged_v7
% make
```

Examples of job scripts are stored in scripts/.
If you want to run larger jobs, modify NX and NY larger (aspect must be 3:1), and make DT defined in cfd.h smaller.
Do, make clean, then again, make.
NOTE: We are sharing a group queue account with all participants of this summer school. So, please specify the CPU time limit less than 5mins for a debug test, and do not submit many jobs at the same time!



Example of Problem Settings (cfd.h)

- For a large test

```
#define NX (360)
#define NY (120)
#define DT  (5e-5)
#define NU  (0.01)
#define END_TIMESTEP       (10000)
#define SAVE_INTERVAL       (200)
```

- For a huge case

```
#define NX (540)
#define NY (180)
#define DT  (2.5e-5)
#define NU  (0.01)
#define END_TIMESTEP       (20000)
#define SAVE_INTERVAL       (400)
```

- For a Challenging case

```
#define NX (2160)
#define NY (720)
#define DT  (2.5e-6)
#define NU  (0.01)
#define END_TIMESTEP
(4000*50)
#define SAVE_INTERVAL
(4000)
```

Challenging case needs 8nodes and 15min elapsed time

Large test

1. **Edit cfd.h** *(available _LARGE_ and disable others)*

```
#define      TEST_CASE   _LARGE_
```

2. **Compile by 'make'.**
3. **Prepare files required to a job**

- **Make a work directory 'run_large' and change directory** to it.

```
% mkdir run_large
% cd run_large
```

- Then, copy **../script/run_large.sh** and the executable ../**solver_fractional** on the directory.

```
% cp ../scripts/run_large.sh .
% cp ../solver_fractional .
```

Large test

4. **Submit a job script**

```
% pjsub run_large.sh
```

5. **Find computed results on the directory.**
   - Data files are stored on the directory, namely,
   - **AVEse_******.dat

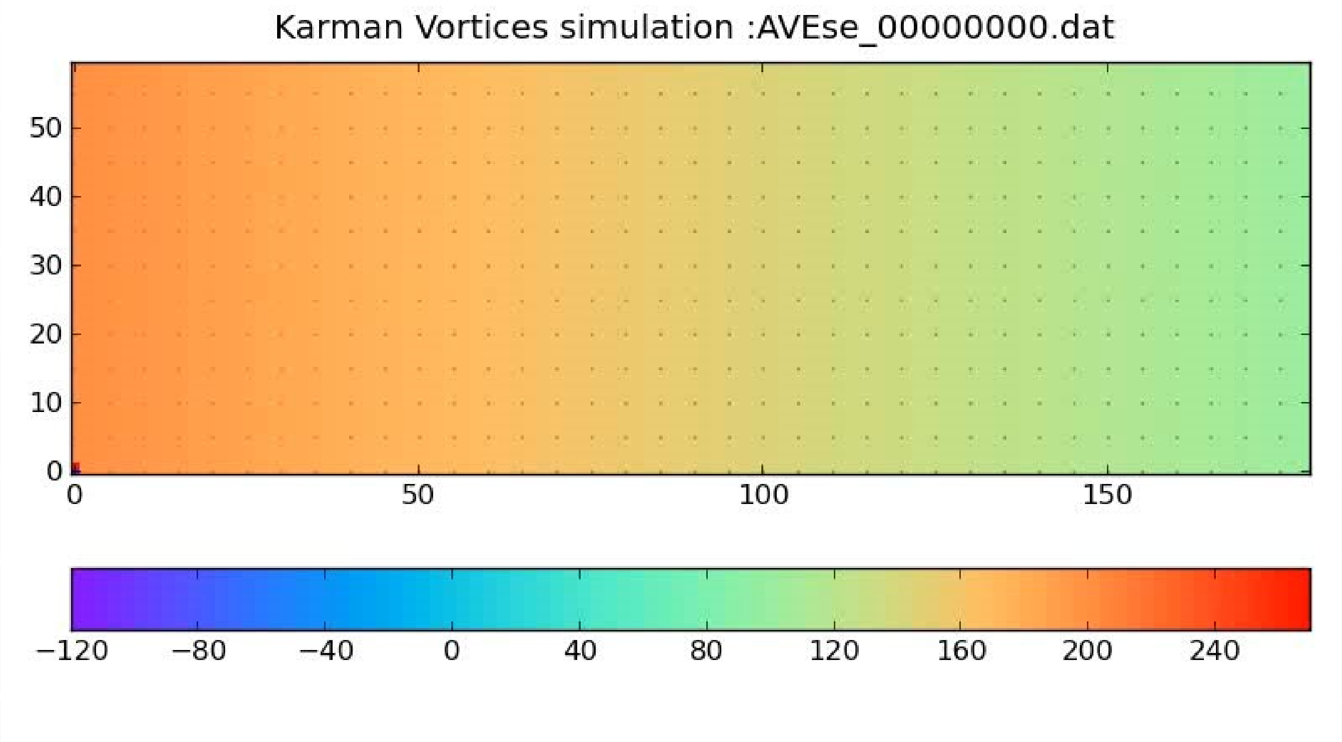
6. **Copy ../view-x.py**
   - Follow the next pages

Visualize a result

- **Use a python script**

Now you are on the work directory.

```
% python3 view-x.py
```


Karman Vortices simulation :AVEse_00000000.dat

view-x.py

- **At the first run, local installation of numpy and matplotlib**
  - **pip3 install --user numpy maplotlib**
- **If you run X-window server on your local PC,**
  - Use view-x.py. Happen a driver problem, check next line commented out or modified correctly
    - **#matplotlib.use('agg')**
- **If your network connection is bad or not using X server on your local PC,**
  - Please edit two lines in view-x.py as

```
#plt.show()
anim.save("plot-z-4.mp4", fps=5)
```

You will get plot-z-4.mp4 after python run, then download it on your local PC.

Viewer on your PC depends on your environment. Probably just double-clicking the file will play the animation.

Hands-on time

Please access **/home/ra020006/l00112/school** **if have not yet copied files.**

PETSc/ → PETSc sample code, a job-script, and Makefile
mpi-petsc_merged_v7.tgz → Karman Vortex CFD codes

Practices

1. **Change the shape of obstacle or put some obstacles in a fluid region**
    1. One square block is already put in the original CFD domain.
    2. **Please modify cfd.h, and main(), consistently.**
    3. **Ghost grid points inside or at the boundary of the obstacle objects must be introduced.**
    4. Visualize as the same way to display the original.
    5. Confirm Karman vortices.

[Cautions]
The objects must have no intersection area.
If the objects are smaller than grid width or almost touches the wall, the CFL condition worsens, then CFD code might diverge. Carefully setup the boundary, and tweak delta-t as a smaller value.

Practices

2. **Use other iterative algorithms**
   1. Current PETSc solver uses a default KSP method
   2. You can select other algorithms by a command-line or add PetscErrorCode KSPSetType(KSP ksp, KSPType type)
   3. Please refer to the list of KSPType
      https://petsc.org/release/docs/manualpages/KSP/KSPType.html#KSPType
   4. Confirm the convergent histories.

Note

- **The CFD code generates a lot of data files.**
  - Quota on our disk space is
    - 5120GiB by this group
    - 1,500,000 i-nodes

  → Unnecessary old files or directories must be compress/zip/tar-ed to reduce the number of i-nodes.

- **CPU time:**
  - If you run the first time, please set up the elapsed time on the job-script smaller, for example, 1min (00:01:00) or lesser.

- **Enjoy the CFD code on Fugaku!**