Third day

Practice for numerical issues

RIKEN CCS HPC Summer School
Toshiyuki Imamura, RIKEN CCS
with assistant Dr. Shuhei Kudo

Numerical Library
What is it? For what?

Numerical Library

Computer simulations create the future

- **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…**
  - FFT, Eigenvalue calculation, SVD, etc…
- **There are reference codes.**
  - They might be examples of good (bad) programming.
- **It provides us with better performance and finer accuracy.**
  - 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 of Numerical Libraries

Computer simulations create the future

- **When you SPEED UP your code, use (probably being linked in most utility programs) the 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

Example of Numerical Libraries

Computer simulations create the future

If you want solve 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)
- GMP big number librart(https://gmplib.org/)

Random number

- QD pack, MPACK, and so on

Multi-precision number

Review back on the CFD code

Core computational part in the CFD code

Computer simulations create the future

- **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, we have the Core 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));
```

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 diverge).
- **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 diverge).
- **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 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 diverge).
- **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 condition).

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

- **GMRES**

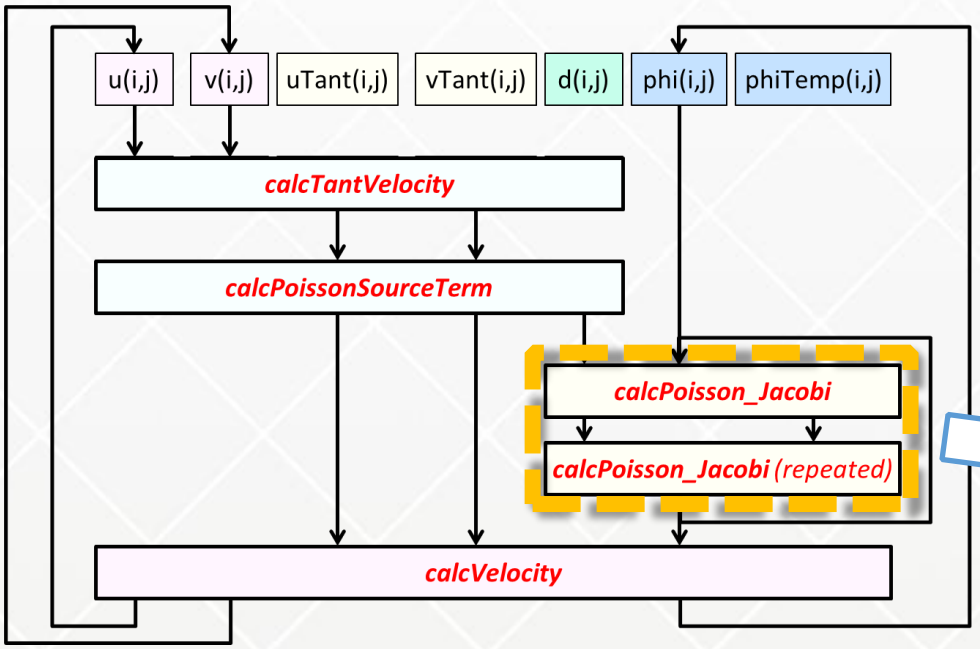
$$\mathrm{argmin}_y\{\|AVy - b\| | V \in \mathcal{K}_A^n(v_0)\}$$

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

Generally, GMRES is known as 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 diverge).
- **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

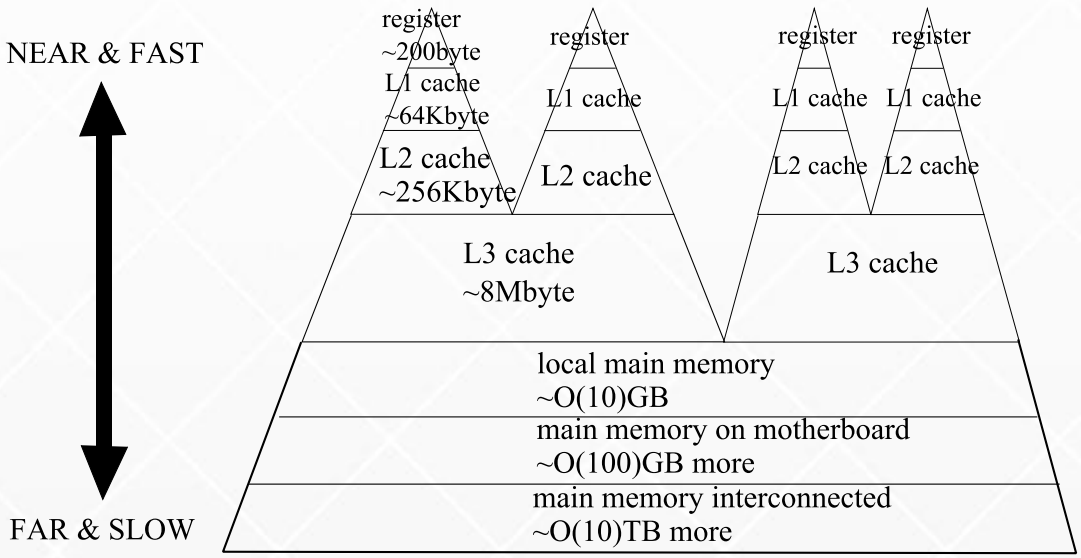
Krylov Subspace method
GMRES, CG, GCR, etc..
with Preconditioners

How does Numerical library achieve high Performance?

How can user utilize it with higher performance?

Difficulties in multicore computing

- **Difficulties in handling multicore.**
  - They are hiding in the deep and complex hierarchy of cache and memory structures.



  - To access the data with a tiny latency, the data must be prefetched on a cache memory, and dealt with carefully. In fact, we should have a view in a mind thoroughly to the cache conflict and memory traffic.
- **Memory bandwidth**
  - Improvement in the memory bandwidth is extremely gradual.
  - It is inevitable to change the software according to such an evolution and complication of hardware.

Balance of flops/memory access

- **BLAS has three categories with respect to the ratio of flops/memory access. Assumption: Data loaded from main memory is stored in a buffer memory on a processor chip and recycle until the procedure completes.**
  - Level 1: vector-vector operations
    - O(N) / O(N) ~O(1)

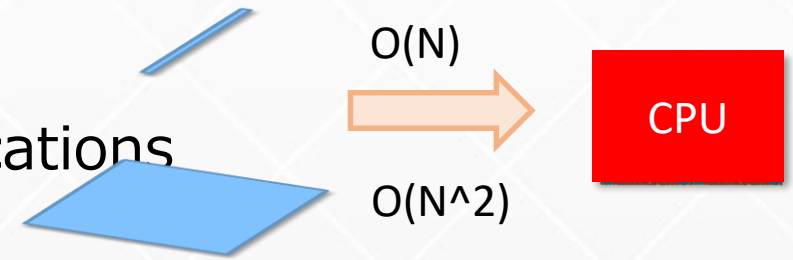
      $y := \alpha x + y$

  - Level 2: Matrix-vector multiplications
    - O(N^2)/O(N^2) ~O(1)

      $y := \alpha Ax + \beta y$

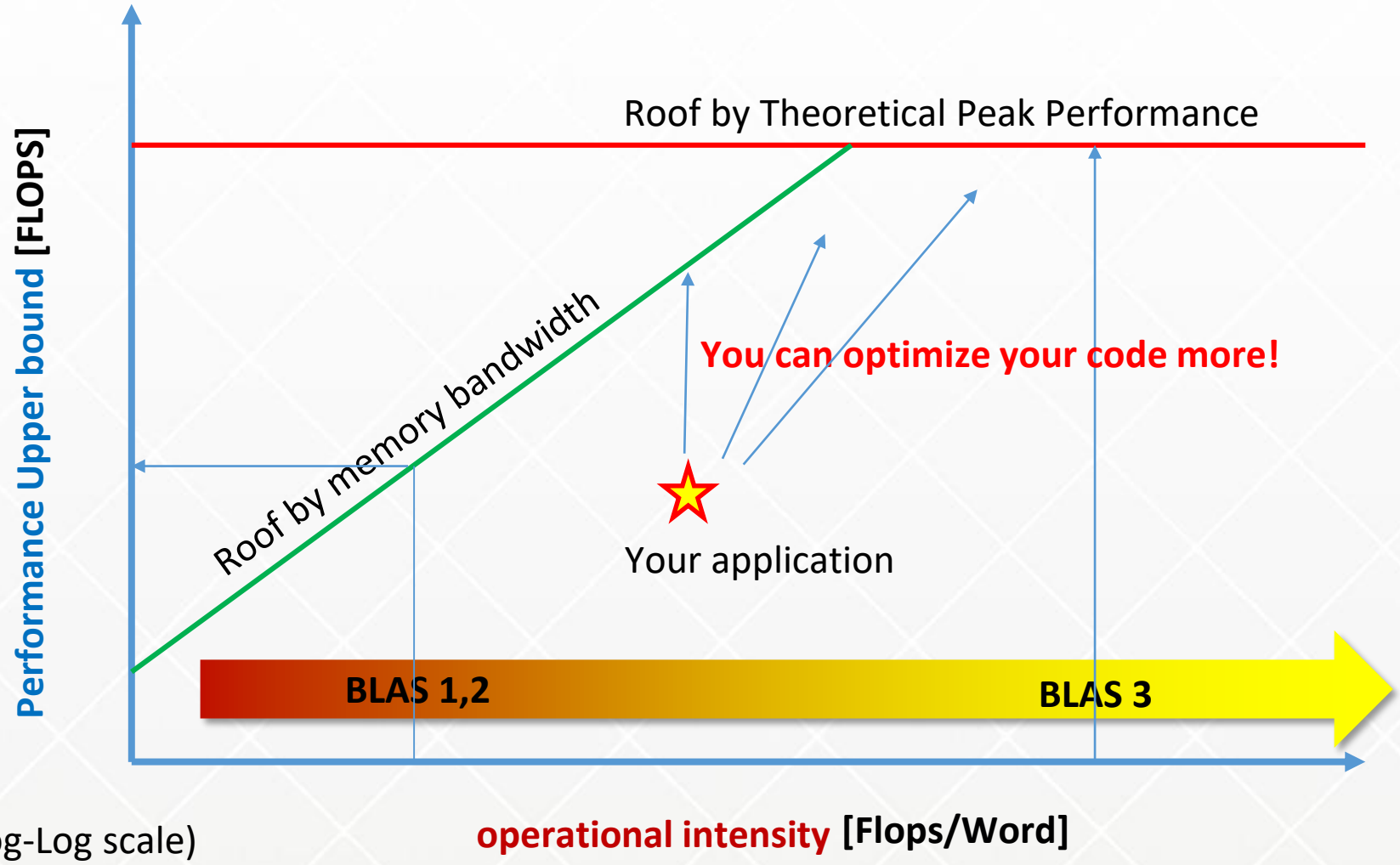
  - Level 3: Matrix-Matrix multiplication
    - O(N^3)/O(N^2) ~O(N) >> O(1)

      $C := \alpha AB + \beta C$



The metric defined by 'flops/Byte' or 'flops/words' is called **'operational intensity'.** It is strongly related to the Roofline model.

Roofline model



(Log-Log scale)

Recent trends of Optimization

Computer simulations create the future

- **Block algorithm**
  - Since block algorithm reduces memory traffic, it leads to larger operational intensity.
  - Re-write your code by using GEMM (or level3 BLAS) calls.
- **Tiling algorithm**
  - It is also one of the block algorithms, but the tiling algorithm reformats the data layout.
    - A(1:NX,1:NY) → A(1:BX,1:BY, 1:NX/BX, 1:NY/BY)

      Submatrix is contiguously stored in memory
  - Contiguous data/stream data can be store into L1/L2 cache.
  - Reduce and avoid TLB-miss trouble

How to use Numerical Libraries

ScaLAPACK, EigenExa, SLEPC

ScaLAPACK

For Dense linear algebra

From the developer site

- **http://www.netlib.org/scalapack/examples/**

Download "sample_pdsyev_call.f".

How to use ScaLAPACK on K

Computer simulations create the future

- **Link on the K computer or other Fujitsu environments**

```
% mpifrtpx -o exe sample_pdsyev_call.f -SCALAPACK -SSL2BLAMP
```

- **Job script**

```
#!/bin/bash
#PJM --rsc-list "rscgrp=small"
#PJM --rsc-list "node=2x2"
#PJM --rsc-list "elapse=00:10:00"
#PJM -S
#PJM --stg-transfiles all
#PJM --mpi "use-rankdir"
#PJM --mpi "proc=4"
#PJM --stgin "rank=* ./exe %r:./"
```

```
. /work/system/Env_base

export OMP_NUM_THREADS=8

date
mpiexec ./exe
date
```

Let's learn the sample code

- **Matrix generator: PDLAMODHILB**
- **Eigensolver: PDSYEV**
- **Output routine: PDLAPRNT**

Other routines are required, such as initializer (BLACS_XXX), finalizer (BLACS_EXIT), and definer of descriptors (DESCINIT), which represent matrix data structures.

BLACS: runtime system that organizes communication on ScaLAPACK.
PDSYEV: QR iteration algorithm
other algorithms are supported in PDSYEVX, PDSYEVD, PDSYEVR.

Matrix format: the 2D block-cyclic distribution and column major ordering

Read the source code

- **PDLAMODHILB**
  - Specialized parallelization or parallel calls are encapsulated in PDELSET routine. **The routine may be called in duplicate manner, however, the owner process only update its local memory** stored in 2D block-cyclic distribution.

```
SUBROUTINE PDLAMODHILB( N, A, IA, JA, DESCA, INFO )
DO 20 J = 1, N
DO 10 I = 1, N
IF( I.EQ.J ) THEN
        CALL PDELSET( A, I, J, DESCA,  ¥
        ( DBLE( N-I+1 ) ) / DBLE( N )+ONE /  ( DBLE( I+J )-ONE ) )
ELSE
        CALL PDELSET( A, I, J, DESCA, ONE / ( DBLE( I+J )-ONE ) )
ENDIF

END
```

EigenExa

One of the R-CCS software for solving a dense eigenvalue problem

How to use EigenExa on K

- **Link on the K computer or other Fujitsu environments**

```
% cp home/ra001016/a03572/SCHOOL/EigenExa/EigenExa-2.4p1-patched-all.tgz .
% tar -zxvf EigenExa-2.4p1-patched-all.tgz
% ./bootstrap; ./configure --host=K
% make
Then cd benchmark/
```

- **Job script**

```
#!/bin/bash
#PJM --rsc-list "rscgrp=small"
#PJM --rsc-list "node=2x2"
#PJM --rsc-list "elapse=00:10:00"
#PJM -S
#PJM --stg-transfiles all
#PJM --mpi "use-rankdir"
#PJM --mpi "proc=4"
#PJM --stgin
"rank=* ./eigenexa_benchmark %r:./“
#PJM --stgin "rank=* ./IN %r:./"

. /work/system/Env_base

export OMP_NUM_THREADS=8

date
mpiexec ./eigenexa_benchmark
date
```

Let's learn the sample code

Computer simulations create the future

- **Matrix generator: mat_set**
- **Eigensolver: eigen_sx**

- **As ScaLAPACK does, initializer (eigen_init) and finalizer (eigen_free) are obligately called in appropriate timing.**
- **EigenExa takes advantage of also 2D cyclic-cyclic distribution (in detail, fixed in NB as one both COL and ROW). Compatible to ScaLAPACK and other utility functions, reciprocally.**

- **Therefore, if descriptor defined with NB=1 and generated by PDLMODHILB can be passed to EigenExa and diagonalized.**

Benchmark code

- **On the benchmark directory,**
  - You can submit a job by `pjsub job-script.sh`.

```
Env_base: K-1.2.0-24
Mon Jul  2 11:07:14 JST 2018
 INPUT FILE='IN'
 ==========================================================
 ## EigenExa version (2.4a) / (May 25, 2017) / (Hanachirusato)
 Solver = eigen_sx / via penta-diagonal format
 Block width =  48 / 128
 NUM.OF.PROCESS= 4 ( 2 2 )
 NUM.OF.THREADS= 8
 Matrix dimension =  1000
 Matrix type =  2  (Random matrix)
 Internally required memory =  8472688  [Byte]
 The number of eigenvectors computed =  1000
 mode 'A' :: all the eigenpairs
 Elapsed time =  0.3770818118937314  [sec]
 FLOP        =  6826626901.333333
 Performance  =  18.10383499286145  [GFLOPS]
 ==========================================================
```

All eigenvalues and corresponding eigenvectors of a randomized matrix N=1000.

Edit an input file

- **Look at 'IN' file, you can edit N and nvec fields.**

```
!
! Input file format
!
! N bx by mode matrix solver
!
! N      : matrix dimension
! nvec   : the number of eigenvetors to be computed
! bx     : block width for the forward transformation
! by     : block width for the backward transformation
! mode   : solver mode { 0 : only eigenvalues }
!                      { 1 : eigenvalues and corresponding eigenv
ectors}
!                      { 2 : mode 1 + accuracy improvement for ei
genvalues}
! matrix : test matrix { 11 types, 0 ... 10 }
!        run with -L option shows the list of test matrices
!        typical ones are as follows
!        { 0 : Frank matrix}
!        { 1 : Toeplitz matrix}
!        { 2 : Random matrix}
! solver : { 0 : eigen_sx, new algorithm, faster on the K }
!        { 1 : eigen_s,  conventional algorithm }
! check_error : { 0 : off, 1 : on }
```

```
!
! if a line starts from '!', the line is treated as a commen
t
!   N  nvec bx  by m t s e
 1000  1000 48 128 1 2 0 0
 1000  1000 48 128 1 2 0 0
 1000  1000 48 128 1 2 0 0
 1000  1000 48 128 1 2 0 0
 1000  1000 48 128 1 2 0 0
-1
```

End mark

Accuracy Check

Random matrix

Mode: solving eigenvalues and vectors

Block factors

N. of eigenvectors to be computed

Matrix dimension

PETSc(SLEPC)

Official site on PETSc and SLEPC

Hands-on exercises

https://www.mcs.anl.gov/petsc/petsc-3.8/src/ksp/ksp/examples/tutorials/

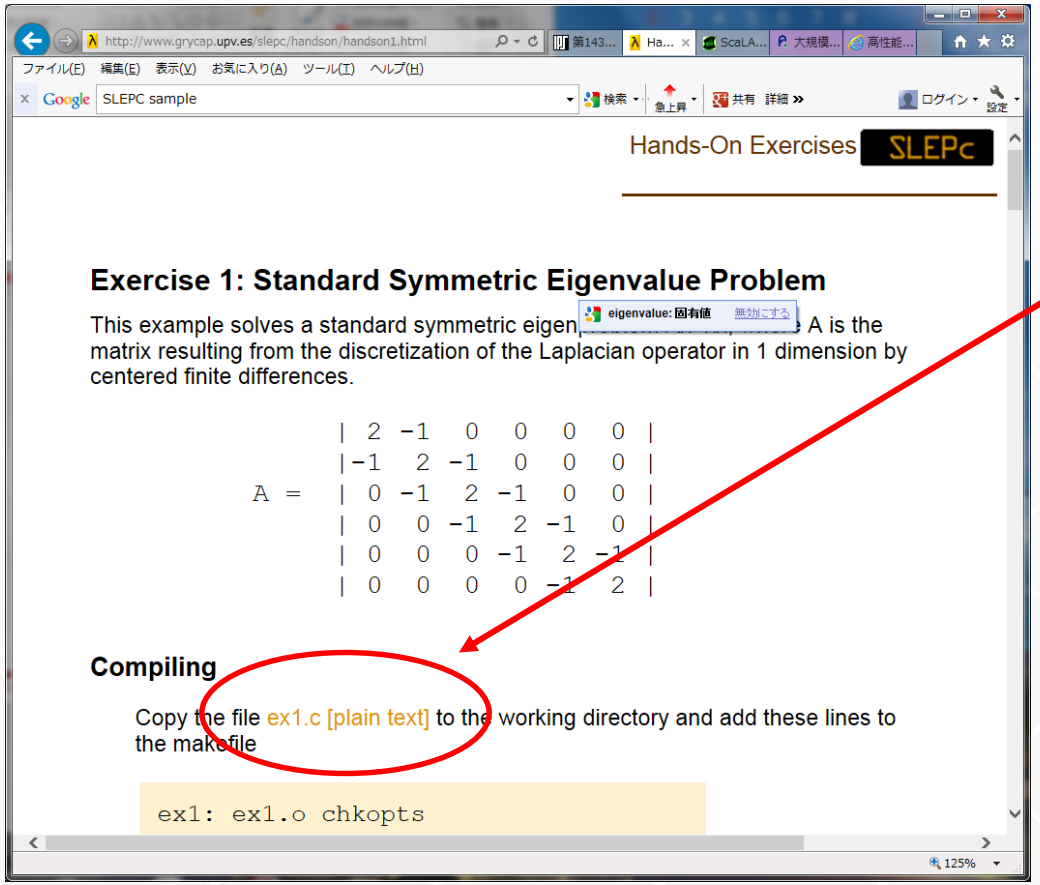
EigenExa | PROJE... | LINKS | 大規模並... | Krylov Meth... | RIKEN Internatio...

https://www.mcs.anl.gov/p

ex29.c: Solves 2D inhomogeneous Laplacian using multigrid
ex30.c: * Example code testing SeqDense matrices with an LDA (leading dimension of the user-allocated arrray) larger than M
ex31.c: Solves 2D compressible Euler
ex32.c: Solves 2D inhomogeneous Laplacian using multigrid
ex34.c: Solves 3D Laplacian using multigrid
ex41.c: Reads a PETSc matrix and vector from a socket connection, solves a linear system and sends the result back
ex42.c: Solves the incompressible, variable viscosity stokes equation in 3d using Q1Q1 elements, ¥n¥
ex43.c: Solves the incompressible, variable viscosity Stokes equation in 2d on the unit domain ¥n¥
ex45.c: Solves 3D Laplacian using multigrid
ex46.c: Solves a linear system in parallel with KSP and DM
ex49.c: Solves the compressible plane strain elasticity equations in 2d on the unit domain using Q1 finite elements
ex50.c: Solves 2D Poisson equation using multigrid
ex51.c: This example solves a linear system in parallel with KSP
ex52.c: Solves a linear system in parallel with KSP
ex53.c: Solves a tridiagonal linear system with KSP
ex54.c: Creates a matrix from quadrilateral finite elements in 2D, Laplacian ¥n¥
ex55.c: 2D, bi-linear quadrilateral (Q1), displacement finite element formulation¥n¥
ex56.c: 3D, tri-linear quadrilateral (Q1), displacement finite element formulation¥n¥
ex58.c: Solves a tridiagonal linear system with KSP
ex62.c: Illustrates use of PCGASM
ex63.cxx:
ex64.c: Illustrates use of the preconditioner GASM
ex65.c: Solves 1D constant coefficient Laplacian using DMSHELL and multigrid
ex66.c: Solves 2D Poisson equation using multigrid
ex67.c: Krylov methods to solve u'' = f in parallel with periodic boundary conditions
ex68.c:
ex69.c:
makefile

ex50.c

http://www.grycap.upv.es/slepc/handson/handson1.html



ex1.c

Compile and link

Makefile

```
ARCH=
PETSC_DIR=/home/ra001016/a03572/petsc-real
PETSC2_DIR=/home/ra001016/a03572/petsc-complex
SLEPC_DIR=/home/ra001016/a03572/slepc

CC=mpifccpx
FC=mpifrtpx
BLAS=-SSL2BLAMP -SCALAPACK

CCFLAGS_PETSC=-Xg -Kfast,openmp -I$(PETSC_DIR)/include
LDFLAGS_PETSC=-L$(PETSC_DIR)/lib -lpetsc $(BLAS)
CCFLAGS_SLEPC=-Xg -Kfast,openmp -I$(PETSC2_DIR)/include -I$(SLEPC_DIR)/include
LDFLAGS_SLEPC=-L$(SLEPC_DIR)/lib -lslepc -L$(PETSC2_DIR)/lib -lpetsc $(BLAS)

all: ksp_exec eps_exec
ksp_exec: ex50.o
	$(CC) $(CCFLAGS_PETSC) -o $@ $< $(LDFLAGS_PETSC)
ex50.o: ex50.c
	$(CC) $(CCFLAGS_PETSC) -c $<

eps_exec: ex1.o
	$(CC) $(CCFLAGS_SLEPC) -o $@ $< $(LDFLAGS_SLEPC)
ex1.o: ex1.c
	$(CC) $(CCFLAGS_SLEPC) -c $<
clean:
	¥rm *.o ksp_exec eps_exec
```

Use **make** command

→

ksp_exec (ex50) and eps_exec are (ex1) generated.

Job submission

```
#!/bin/bash
#PJM --rsc-list "rscgrp=small"
#PJM --rsc-list "node=4"
#PJM --rsc-list "elapse=00:10:00"
#PJM -S
#PJM --stg-transfiles all
#PJM --mpi "use-rankdir"
#PJM --mpi "proc=4"
#PJM --stgin "rank=* ./eps_exec %r:./“

. /work/system/Env_base
export OMP_NUM_THREADS=8

mpiexec ./eps_exec -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,**

```
$ ./eps_exec -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

```
$ ./eps_exec -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

```
$ ./eps_exec -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)

Computer simulations create the future

```
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 )
```

How to setup a matrix? (in fortran90)

Computer simulations create the future

- **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, ierr )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

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

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

How to setup a matrix? (in fortran90)

Computer simulations create the future

- **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, ierr )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

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

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

Create a matrix handler

How to setup a matrix? (in fortran90)

Computer simulations create the future

- **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, ierr )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

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

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

Matrix size MxN

How to setup a matrix? (in fortran90)

Computer simulations create the future

- **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, ierr )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

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

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

For a mxn block matrix, array values are set.

How to setup a matrix? (in fortran90)

Computer simulations create the future

- **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, ierr )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, N, ierr )

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

MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
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

Computer simulations create the future

- **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: 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

Computer simulations create the future

- **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.

- Prepare 3 call-back and 1 utility functions;
  - **ComputeRHS,**
  - **ComputeStencil,**
  - **SetInitialGuess,** and
  - **GetComputedResult.**

- Then, associate them with a DM handler.
- KSPSolve() organizes RHS and MatVec, etc.

Use a PETSc KSP solver

Computer simulations create the future

- **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.

- Prepare 3 call-back and 1 utility functions;
  - **ComputeRHS,**
  - **ComputeStencil,**
  - **SetInitialGuess,** and
  - **GetComputedResult.**

- Then, associate them with a DM handler.
- KSPSolve() organizes RHS and MatVec, etc.

Use a PETSc KSP solver

Computer simulations create the future

- **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.

- Prepare 3 call-back and 1 utility functions;
  - **ComputeRHS,**
  - **ComputeStencil,**
  - **SetInitialGuess,** and
  - **GetComputedResult.**

- Then, associate them with a DM handler.
- KSPSolve() organizes RHS and MatVec, etc.

Use a PETSc KSP solver

Computer simulations create the future

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

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

   - 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

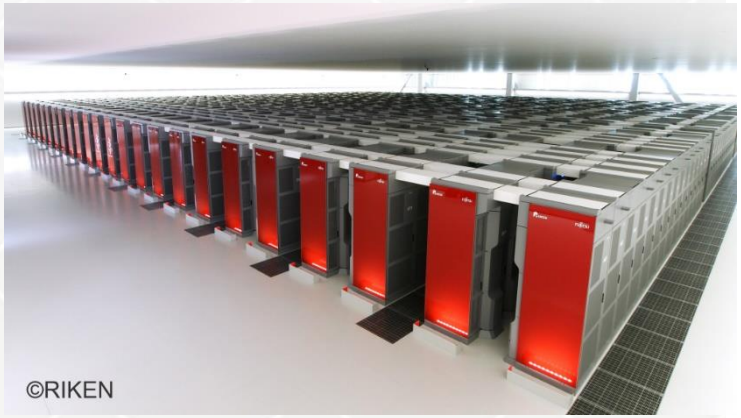
4. **GetComputedResult.**
   - Retrieve the result computed by using PETS.

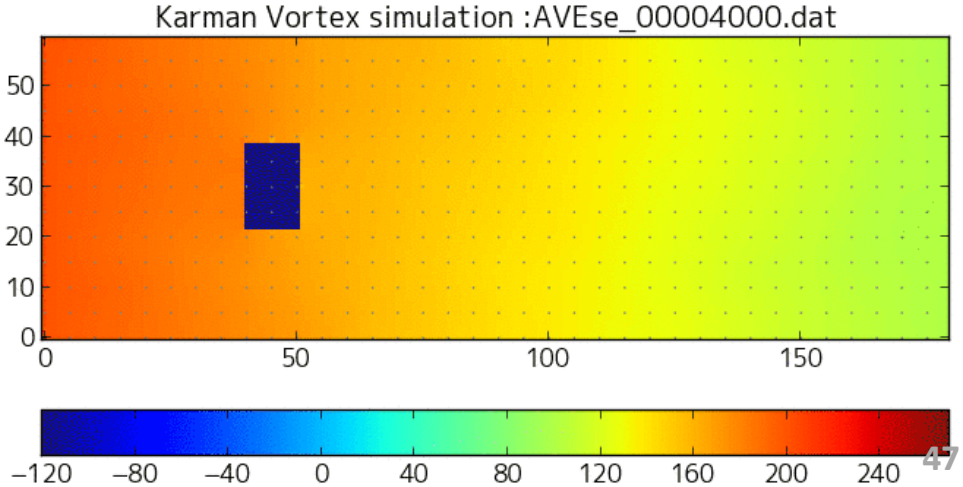
Let's Run the CFD code on K computer

Computer simulations create the future

```
% cp /home/ra001016/a03572/SCHOOL/CFD/mpi-petsc_merged_v4.tgz .
% tar zxvf mpi-petsc_merged_v4.tgz
% cd mpi-petsc_merged_v4
% 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 smaller DT defined in cfd.h. Do, make clean, then again, make.
NOTE: We are sharing a CPU group account. So, please specify the CPU time limit less than 30min so as not to waste CPU time!





Example of Problem Settings (cfd.h)

Computer simulations create the future

- **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 (2700)
#define NY (900)
#define DT  (1e-6)
#define NU  (0.01)
#define END_TIMESTEP
(10000*3)
#define SAVE_INTERVAL
(10000)
```

Large test

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

   #define TEST_CASE _LARGE_

2. **Compile by 'make'.**
3. **Prepare files required to a job** *(in case of 48 nodes)*

   - Make a work directory and change directory to it.
   - Copy **../script/run_batch_48mpi.sh** and the executable ../**solver_fractional** on the directory.

4. **Submit a job script**

   % pjsub run_batch_48mpi.sh

5. **Find and visualize a result on the directory.**

   - Data files are packed into a tar file.

   % tar -zxvf AVEse_data_48.tgz

   - You may also find **run_batch_48mpi.sh.oXXXXX**.

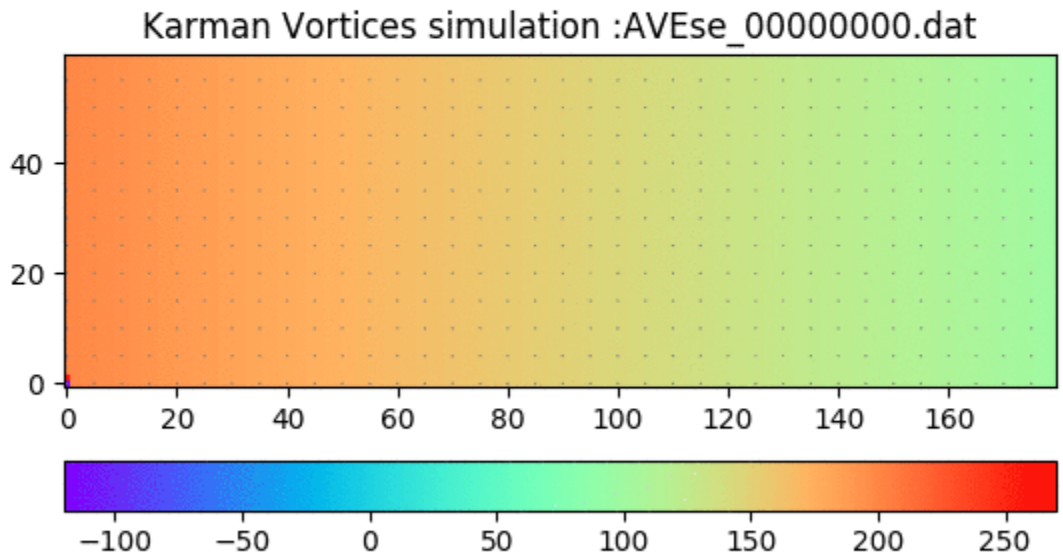
Visualize a result

- **Use a python script**

Now you are on the work directory.

```
% tar -zxvf AVEse_data_48.tgz
% python2.7 view.py
```

Huge case computed with 48 nodes



Hands-on time

Please access /home/ra001016/a03572/SCHOOL

ScaLAPACK/ → a sample code, a jobscript and Makefile
EigenExa/ → a sample code, a jobscript and Makefile
PETSc/ → PETSc sample codes, a jobscript and Makefile
CFD/ → Karman Vortex CFD codes

Basic hand-on works

Computer simulations create the future

BASE:

1. Run a sample code provided by the official developer site of ScaLAPACK.
2. Run a PETSc/SLEPc sample codes.
3. Confirm the CFD code with 'large' or 'huge' cases.

PRACTICAL:

For ScaLAPACK practices,

1. Modify **PDLAMODHILB**, then solve a real symmetric eigenvalue problem as you define or choose freely.
2. Also, vary the matrix dimension **N,** which is fixed in a source code, and the number of processes （**NPROW, NPCOL**）, then confirm parallel efficiency!
3. If you want to move advanced problem, try another routine pdsyevd!

Additional hands-on works

Computer simulations create the future

MORE ADVANCED:
For CFD practices,

1. **Try the challenging parameter case of the CFD code!** Please do not forget the CPU time limitation specified in the job script. Since to visualize Karman vortices takes a lot of iteration (delta T is too small), just stop the run at a couple of hundred iterations.
2. **Compare the two cases if PETSc uses or not.** You can switch them by activating the next macro in cfd.c (1: PETSC, 0: non-PETSC). Also, be aware of the CPU time, because the Jacobi iterative algorithm often fails in case of a challenging problem.

```
#defined USE_PETSC 1
```

MORE^2 ADVANCED:
(See next pages) Compute 9 eigenmodes of a Elastic plate analysis whose eigenvalues are from the largest by using SLEPSc/PETSc, and visualize the corresponding eigenvectors as a set of m x m images.

Advanced Lesson

- **Governing equation of vibration of an Elastic rectangular plate:**
  - Bi-harmonic equation

$$D\left(\frac{\partial^4 u}{\partial x^4}+2\frac{\partial^4 u}{\partial x^2\partial y^2}+\frac{\partial^4 u}{\partial y^4}\right)+\nu\frac{\partial^2 u}{\partial t^2}=0$$

$$D=\frac{Eh^3}{12(1-\mu^2)},\ \ \nu=\rho h$$

- **(E: Young ratio, h: width, ρ: density, μ: Poisson ratio)**
- **By using the variable separation method, we obtain**

$$\frac{\partial^4 u}{\partial x^4}+2\frac{\partial^4 u}{\partial x^2\partial y^2}+\frac{\partial^4 u}{\partial y^4}=\lambda u$$ **: eigenvalue problem**

Try then!

- **Boundary (Dirichlet)+Clamped Edge(Neumann) → Discretization**

$$u(\Gamma) = 0 \qquad \frac{\partial u}{\partial n}(\Gamma) = 0 \quad n : \text{normal} \quad \text{direction}$$

- **Using Kronecker tensor product notation, we present**

$$A = \frac{1}{h_x^4}(I_{N_y-1} \otimes X_{N_x-1}) + \frac{2}{h_x^2 h_y^2}(Y_{N_y-1} \otimes Y_{N_x-1}) + \frac{1}{h_y^4}(X_{N_y-1} \otimes I_{N_x-1})$$

$$X_m = \begin{bmatrix} 7 & -4 & 1 & & & \\ -4 & 6 & -4 & 1 & & \\ & & & & & \\ & & 1 & -4 & 6 & -4 \\ & & & 1 & -4 & 7 \end{bmatrix}_{m \times m} \qquad Y_m = \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & & & & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}_{m \times m}$$

Try then!

- **For the simplicity, the domain to be computed is fixed as a square shape and other parameters are supposed to be Nx=Ny=m, hx=hy=1.**

$$A = \begin{bmatrix} X_m & & & \\ & X_m & & \\ & & \ddots & \\ & & & X_m \end{bmatrix} + 2 \begin{bmatrix} -2Y_m & Y_m & & & \\ Y_m & -2Y_m & Y_m & & \\ & & & \ddots & \\ & & & Y_m & -2Y_m \end{bmatrix}$$

$$+ \begin{bmatrix} 7I & -4I & I & & \\ -4I & 6I & -4I & I & \\ & & & \ddots & \\ & & I & -4I & 7I \end{bmatrix}$$