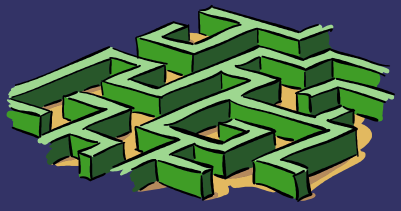
E-science

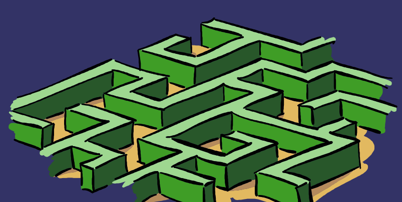
Portable, Extensible Toolkit for Scientific Computation (PETSc) - functionality and experience with problems from Biomechanics

Guanwen Ying



Content

- Introduction to Petsc
- Configure build and install
- Matrices vectors, solvers and preconditioners
- Application of trabecular bone problem
- A small Petsc example



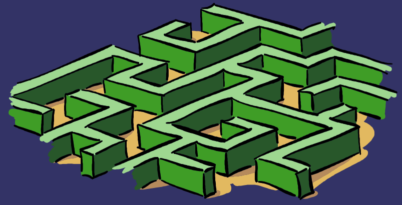
Introduction

- What is Petsc?

  The Portable, Extensible Toolkit for Scientific Computation (PETSc) eases the development of large-scale scientific application codes, particularly the numerical solution of PDEs.

  Developed at the Mathematics and Computer Science Division of Argonne National Laboratory.

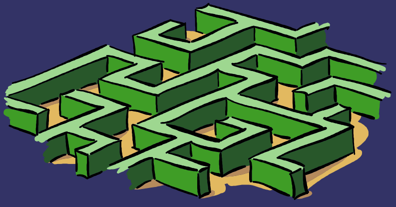
- http://www.mcs.anl.gov/petsc/petsc-as/index.html



Introduction

Features of Petsc

- Vectors and matrices (sparse/dense)
- Krylov space method
- Preconditioners
- Graphic devices
- Parallel support for sparse matrix, some solvers and preconditioners
- Make good use of BLAS/LAPACK routine



Introduction

System requirement

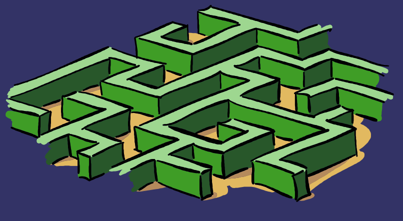
- Any parallel system supporting MPI
- PC running linux or windows (Cygwin is necessary for windows)

Advantage of Petsc

- Easy for beginner
- Almost fully supported by the developer
- Various interface to other packages

Disadvantage of Petsc

- No load balancing
- Has to work on a low latency network



Introduction

PDE Solver

TS(time stepping)

SNES
(nonlinear equation solvers)

SLES
(linear equation solvers)

KSP
(Krylov space method)

Preconditioner

Draw

External Packages

Matrices

Vectors

Index Set

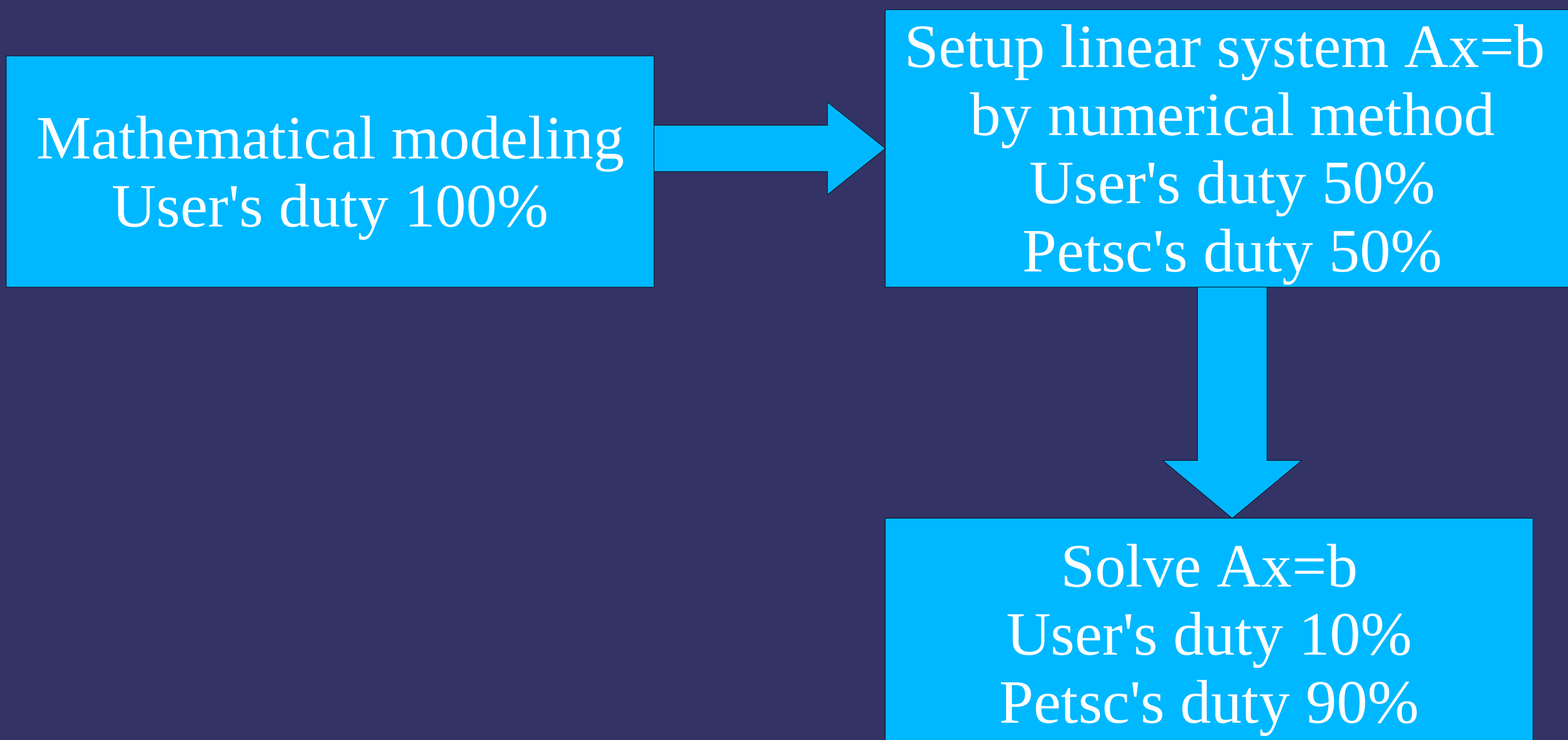
BLAS

LAPACK

MPI

Introduction

Petsc can play almost the same role as Matlab



Configure build and run

Set environment variables

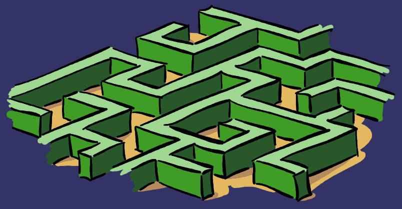
- export PETSC_DIR='.......'
- export PETSC_ARCH='.......'

Running configuration

- ./configure

Specify BLAS/LAPACK/MPI directory(optional)

- --with-blas-lapack-dir= '.........' --with-mpi-dir='.......'



Configure build and run

Specify compilers(optional)

- --with-cc=gcc --with-fc=gfortran

Specify external package to download(optional)

- --download-hypre=yes

Need help?

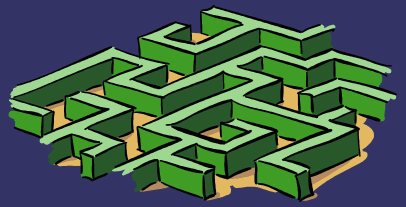
- --help

Compile Petsc on your system

- make all

Test run

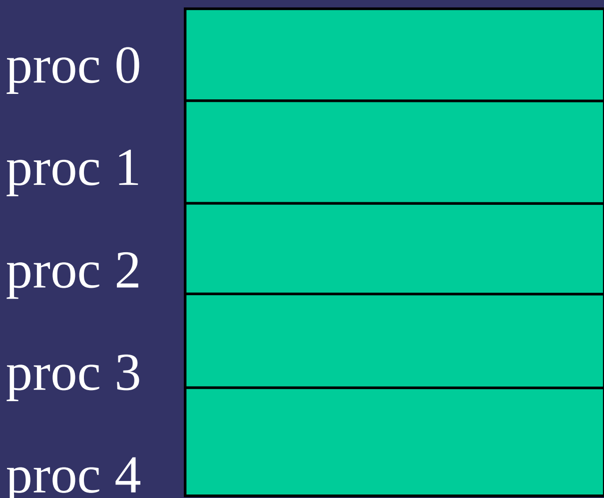
- Make test



Matrices vectors

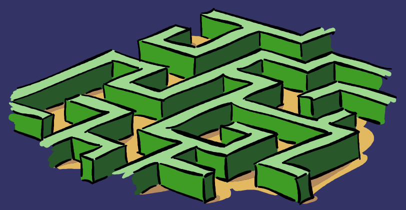
Matrix types (mostly used)

- Sequential AIJ, Sequential Block AIJ (for single processor)
- MPI AIJ, MPI Block AIJ (for more than one processor)

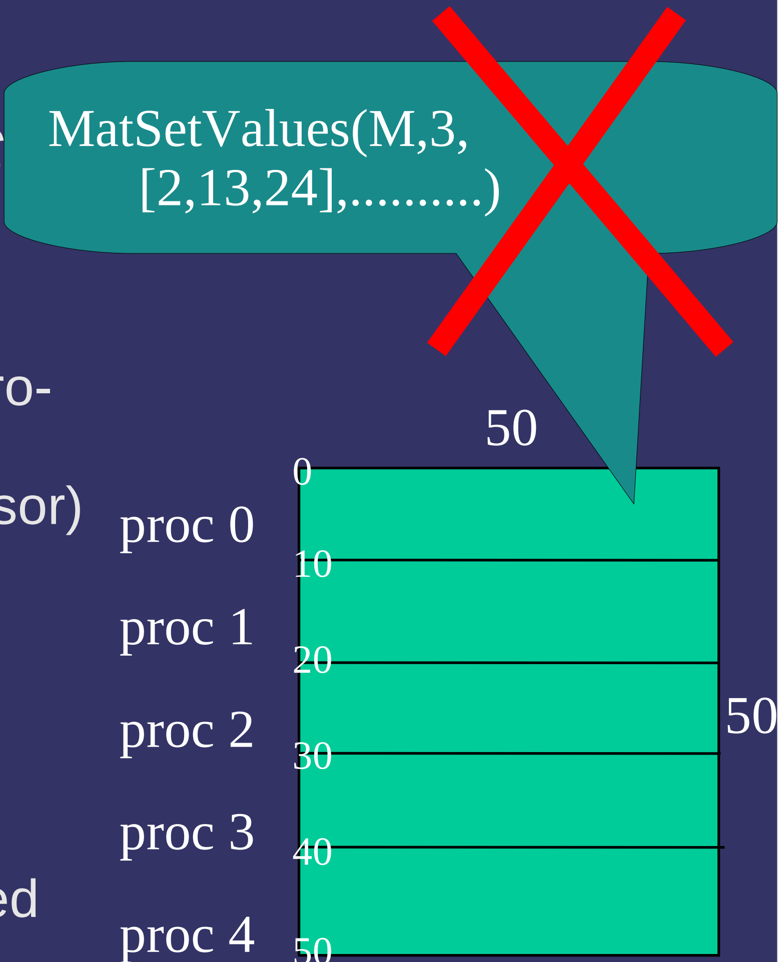


Matrix Create

- MatCreate(MPI_Comm comm, Mat *M);
- MatSetSizes(Mat M,PetscInt row,PetscInt column,PetscInt ROW,PetscInt COLUMN);
- MatSetFromOptions(Mat M);
- Mat****SetPreallocation(.........) (optional, can speed up matrix assembly for large matrix)
- MatGetOwnershipRange(Mat M, PetscInt *Istart, PetscInt *Iend);
- MatSetValues(Mat M,PetscInt m,const PetscInt idxm[],PetscInt n,const PetscInt idxn[],const PetscScalar v[],InsertMode addv)
- MatAssemblyBegin(Mat M,MAT_FINAL_ASSEMBLY);
- MatAssemblyEnd(Mat M,MAT_FINAL_ASSEMBLY);



Matrices vectors

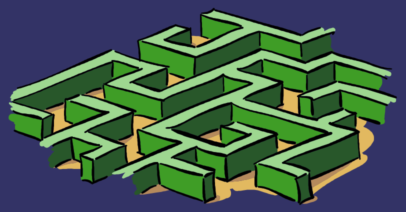


Matrix types (mostly used)

- Sequential AIJ, Sequential Block AIJ (for single processor)
- MPI AIJ, MPI Block AIJ (for more than one processor)

Matrix Create

- MatCreate(MPI_Comm comm, Mat *M);
- MatSetSizes(Mat M,PetscInt row,PetscInt column,PetscInt ROW,PetscInt COLUMN);
- MatSetFromOptions(Mat M);
- Mat****SetPreallocation(.........) (optional, can speed up matrix assembly for large matrix)
- MatGetOwnershipRange(Mat M, PetscInt *Istart, PetscInt *Iend);
- MatSetValues(Mat M,PetscInt m,const PetscInt idxm[],PetscInt n,const PetscInt idxn[],const PetscScalar v[],InsertMode addv)
- MatAssemblyBegin(Mat M,MAT_FINAL_ASSEMBLY);
- MatAssemblyEnd(Mat M,MAT_FINAL_ASSEMBLY);



Matrices vectors

Preallocate memory for matrix

- MatSeqAIJSetPreallocation(Mat M,PetscInt nz_row,PetscInt *nz_table)
- MatMPIAIJSetPreallocation(Mat M,PetscInt diag_nz,PetscInt *diag_table,PetscInt offdiag_nz,PetscInt *offdiag_table)
- Example

```
           1  2  0  |  0  3  0  |  0  4
Proc0      0  5  6  |  7  0  0  |  8  0
           9  0 10  | 11  0  0  | 12  0
-------------------------------------
          13  0 14  | 15 16 17  |  0  0
Proc1      0 18  0  | 19 20 21  |  0  0
           0  0  0  | 22 23  0  | 24  0
-------------------------------------
Proc2     25 26 27  |  0  0 28  | 29  0
          30  0  0  | 31 32 33  |  0 34
```

- MatSeqAIJSetPreallocation(M, 5, PETSC_NULL)
- MatSeqAIJSetPreallocation(M, PETSC_NULL,[4 4 4 5 4 3 5 5])
- MatMPIAIJSetPreallocation(M, 3, PETSC_NULL,4,PETSC_NULL)
- MatMPIAIJSetPreallocation(M,PETSC_NULL,[2 3 1],PETSC_NULL,[2 2 4])

Matrices vectors

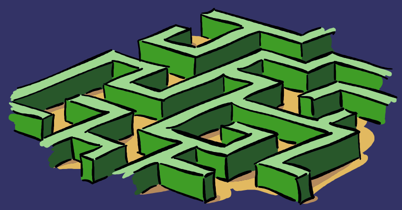
Preallocate memory for matrix

- MatSeqAIJSetPreallocation(Mat M,PetscInt nz_row,PetscInt *nz_table)
- MatMPIAIJSetPreallocation(Mat M,PetscInt diag_nz,PetscInt *diag_table,PetscInt offdiag_nz,PetscInt *offdiag_table)
- Example

```
         1  2  0  |  0  3  0  |  0  4
Proc0    0  5  6  |  7  0  0  |  8  0
         9  0 10  | 11  0  0  | 12  0
-------------------------------------
        13  0 14  | 15 16 17  |  0  0
Proc1    0 18  0  | 19 20 21  |  0  0
         0  0  0  | 22 23  0  | 24  0
-------------------------------------
Proc2   25 26 27  |  0  0 28  | 29  0
        30  0  0  | 31 32 33  |  0 34
```

Most these methods would waste memory

- MatSeqAIJSetPreallocation(M, 5, PETSC_NULL)
- MatSeqAIJSetPreallocation(M, PETSC_NULL,[4 4 4 5 4 3 5 5])
- MatMPIAIJSetPreallocation(M, 3, PETSC_NULL,4,PETSC_NULL)
- MatMPIAIJSetPreallocation(M,PETSC_NULL,[2 3 1],PETSC_NULL,[2 2 4])



Matrix vector types

Vector create

- VecCreate(MPI_Comm comm, Vec *vec)
- VecSetSizes(Vec v, PetscInt n_local, PetscInt n_global);
- VecSetFromOptions(Vec vec);
- VecSetValue(Vec v,int row,PetscScalar value, InsertMode mode);
- VecAssemblyBegin(Vec vec);
- VecAssemblyEnd(Vec vec);

Matrix vector operations

Set one value at a time

- MatSetValue(Mat M,PetscInt row, PetscInt column,PetscScalar value[],InsertMode addv)
- VecSetValue(Vec v,int row,PetscScalar value, InsertMode mode);

Set more than one value at a time

- MatSetValues(Mat mat,PetscInt m,const PetscInt idxm[],PetscInt n,const PetscInt idxn[],const PetscScalar v[],InsertMode addv)
- VecSetValues(Vec x,PetscInt ni,const PetscInt ix[],const PetscScalar y[],InsertMode iora)

Linear solvers and preconditioners

Krylov space method

- KSPCreate(MPI_Comm comm,KSP *ksp)
- KSPSetOperators(KSP ksp,Mat Amat,Mat Pmat,MatStructure flag)
- KSPSetType(KSP ksp, const KSPType type)
- PCCreate(MPI_Comm comm,PC *pc)
- PCSetType(PC pc,const PCType type)
- KSPSolve(KSP ksp,Vec b,Vec x)

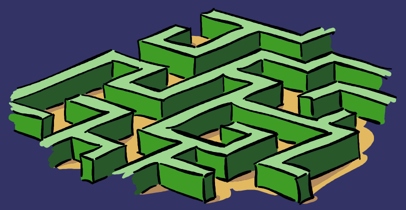
Or specified by runtime options

- KSPSetFromOptions(KSP ksp)
- PCSetFromOptions(PC pc)

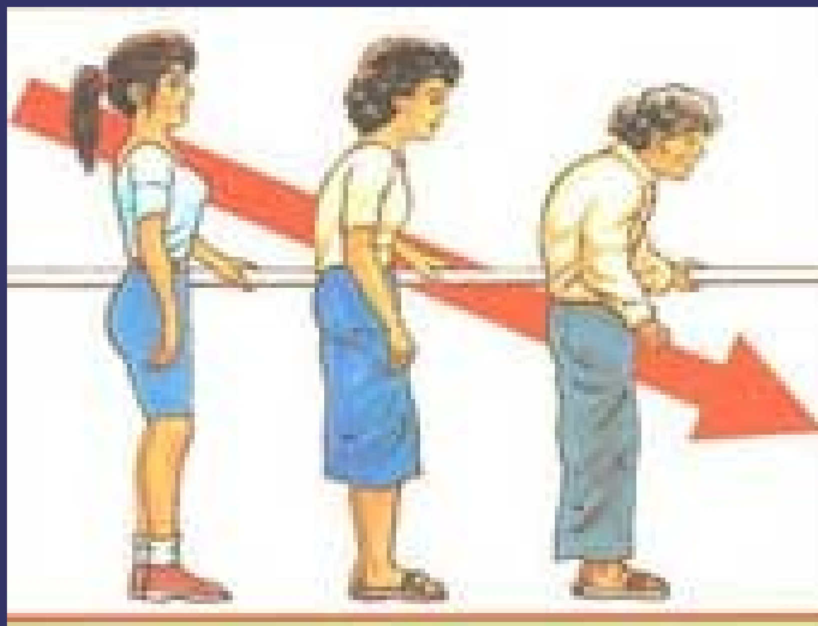
Make file

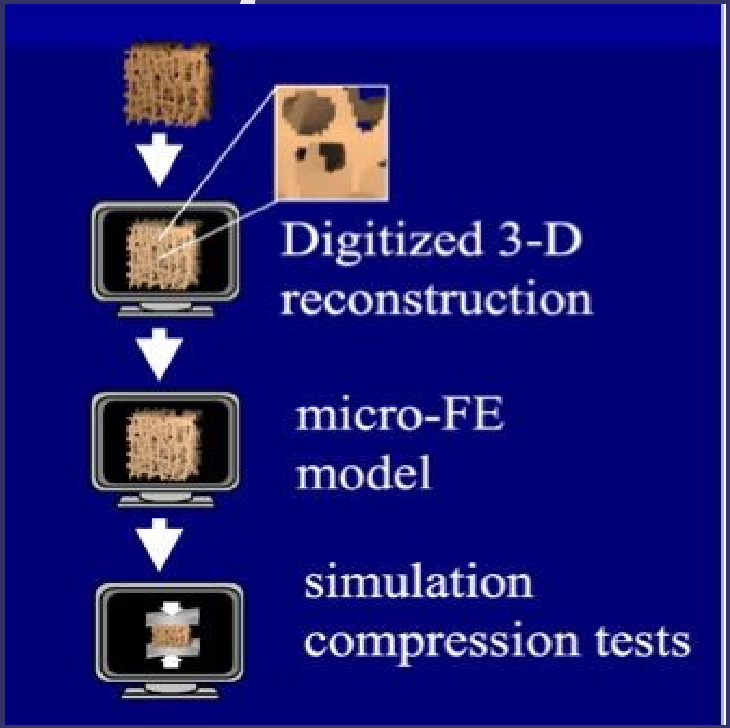
- include ${PETSC_DIR}/conf/base
- example: example.o -----same name as source file
- -${CLINKER} -o example example.o ${PETSC_LIB}
- ${RM} *.o

- This makefile style make it possible for Petsc to link other shared libraries (e.g. free glut, OpenGL).

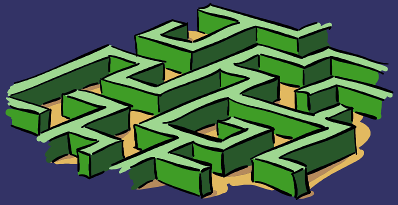


Application of bone problem



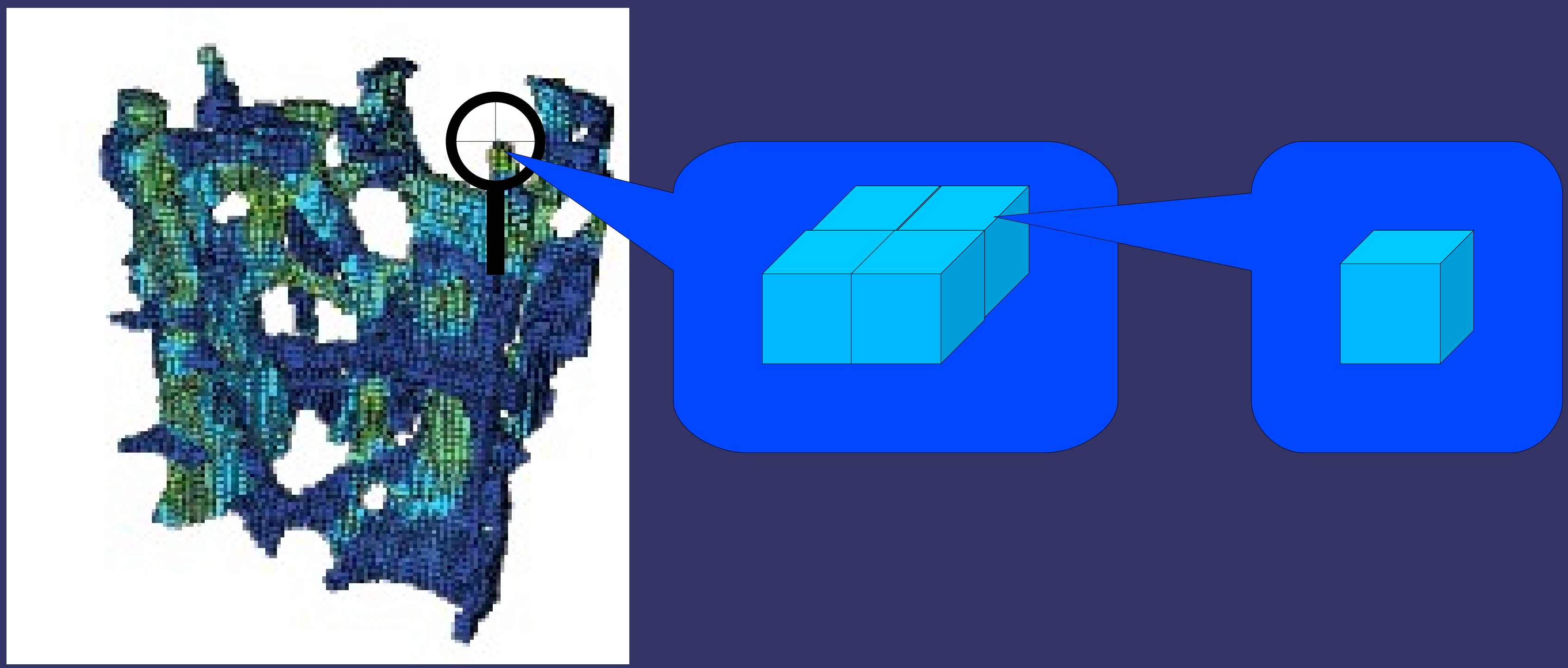


- Background is a disease of "osteoporo-sis"
- Simulate the human bone structure with micro finite element method



Application of bone problem

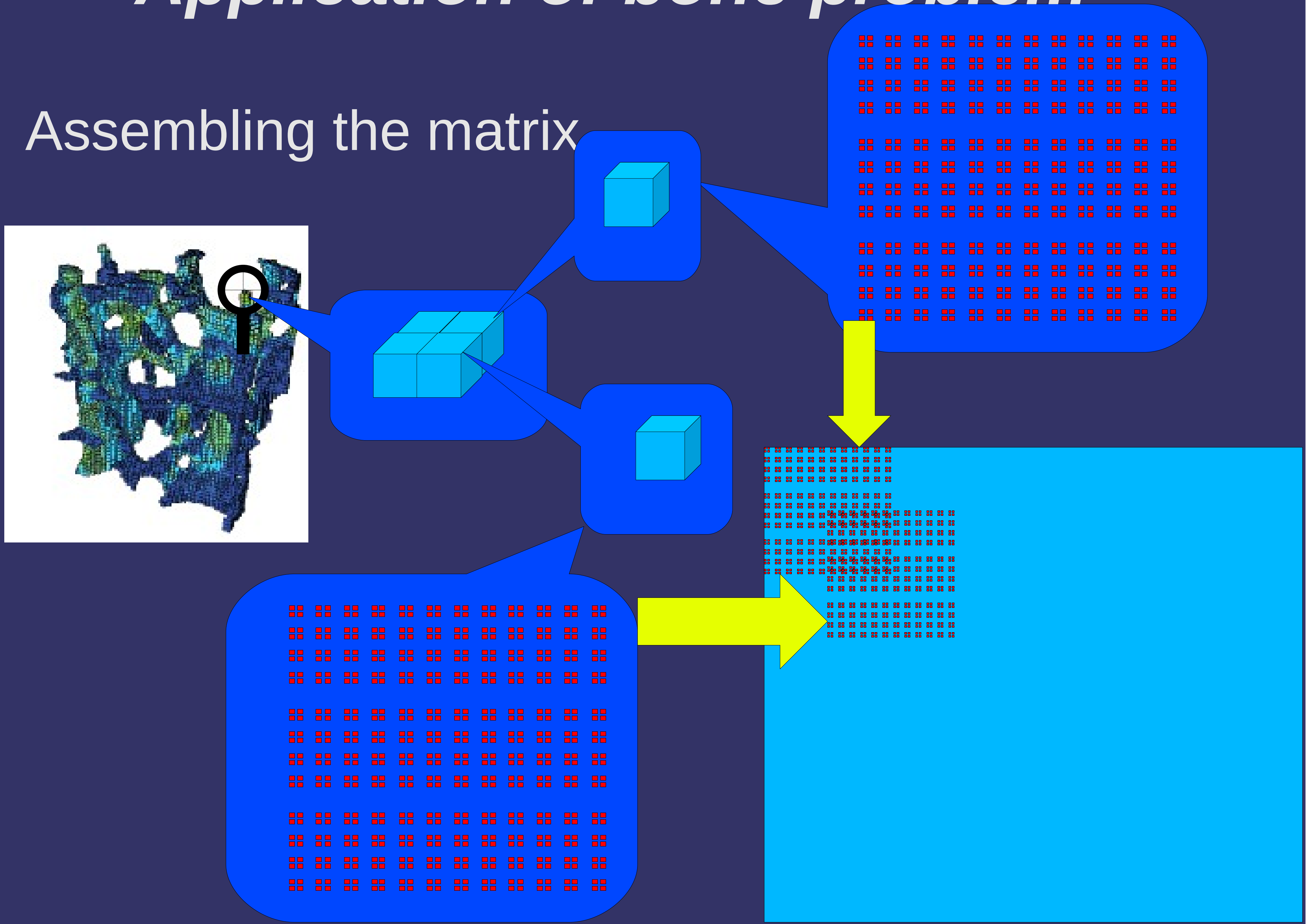
- Element type



- Cube element, eight nodes per element

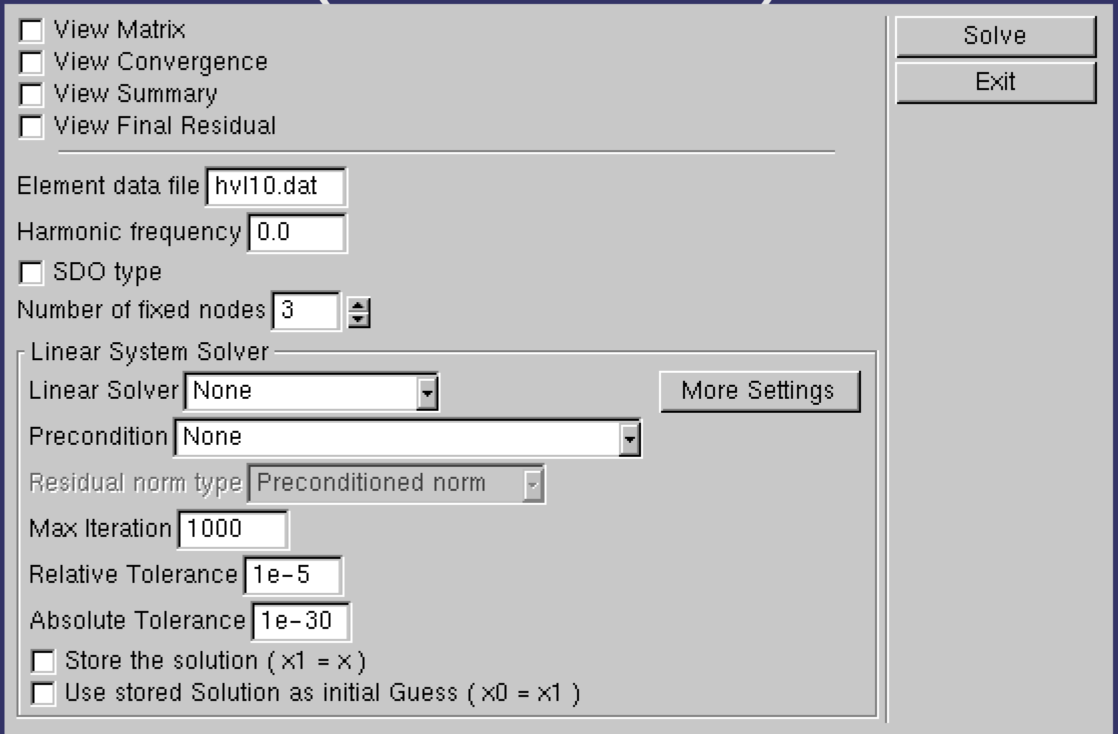
Application of bone problem

Assembling the matrix



Application of bone problem

- Make use of Petsc's Krylov space method
- Use Petsc together with other programing li-brary
- User interface
- Glut programming to display geometry and solution (future work)



Small example

- 1D problem

$$-u''(x)=f(x) \quad 0<x<1$$
$$u(0)=u(1)=0, f(x)=\sin(x)$$

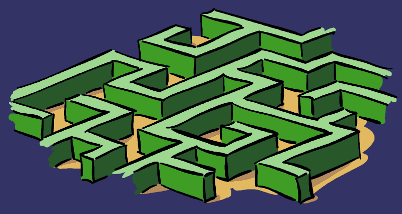
- Discretization

$$x_j = jh \quad j=1,2,\ldots n$$
$$-D_+ D_- u_j = f_j$$

- Linear system

$$Au=f$$

$$A=\frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ .. & .. & .. & & \\ & .. & .. & .. & \\ 0 & & & -1 & 2 \end{bmatrix} \quad u=\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ .. \\ .. \\ u_n \end{bmatrix} \quad f=\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ .. \\ .. \\ f_n \end{bmatrix}$$



Small example

- Code comparison with MATLAB
- Example usage of additional development library (glut/glui)

The End

Thank You