Introduction to PETSc
KSP, PC

CS595, Fall 2010
Linear Solution

Main Routine

PETSc

Linear Solvers (KSP)

PC

Evaluation of $A$ and $b$

Application Initialization

Post-Processing

User code

PETSc code

solutions:

linear
Creating the KSP Context

- **C/C++ version**
  
  ```c
  ierr = KSPCreate(PETSC_COMM_WORLD, &ksp);
  ```

- **Fortran version**
  
  ```fortran
  call KSPCreate(PETSC_COMM_WORLD, ksp, ierr)
  ```

- Provides an **identical** user interface for all linear solvers
  - uniprocess and parallel
  - real and complex numbers
Context Variables

- Are the key to solver organization
- Contain the complete state of an algorithm, including
  - **parameters** (e.g., convergence tolerance)
  - **functions** that run the algorithm (e.g., convergence monitoring routine)
  - **information** about the current state (e.g., iteration number)
KSP Structure

• Each KSP object actually contains two parts:
  – **Krylov Space Method**
    • The iterative method
    • The context contains information on method parameters (e.g., GMRES search directions), work spaces, etc
  – **PC — Preconditioners**
    • Knows how to apply a preconditioner
    • The context contains information on the preconditioner, such as what routine to call to apply it
Linear Solvers in PETSc

**Krylov Methods (KSP)**
- Conjugate Gradient
- GMRES
- CG-Squared
- Bi-CG-stab
- Transpose-free QMR
- etc.

**Preconditioners (PC)**
- Block Jacobi
- Overlapping Additive Schwarz
- ICC, ILU (sequential only)
- ILU(k), LU (direct solve, sequential only)
- Arbitrary matrix
- etc.

**beginner**
Basic Linear Solver Code (C/C++)

KSP ksp; /* linear solver context */
Mat A; /* matrix */
Vec x, b; /* solution, RHS vectors */
int n, its; /* problem dimension, number of iterations */

MatCreate(PETSC_COMM_WORLD,PETSC_DECIDE,PETSC_DECIDE,n,n,&A);
MatSetFromOptions(A);
/* (code to assemble matrix not shown) */
VecCreate(PETSC_COMM_WORLD,&x);
VecSetSizes(x,PETSC_DECIDE, n);
VecSetFromOptions(x);
VecDuplicate(x,&b);
/* (code to assemble RHS vector not shown) */

KSPCreate(PETSC_COMM_WORLD, &ksp);
KSPSetOperators(ksp, A, A, DIFFERENT_NONZERO_PATTERN);
KSPSetFromOptions(ksp);
KSPSolve(ksp, b, x);
KSPDestroy(ksp);

beginner

Indicate whether the preconditioner has the same nonzero pattern as the matrix each time a system is solved. This default works with all preconditioners. Other values (e.g., SAME_NONZERO_PATTERN) can be used for particular preconditioners. Ignored when solving only one system.

solvers: linear
Basic Linear Solver Code (Fortran)

KPS    ksp
Mat    A
Vec    x, b
integer n, its, ierr

call  MatCreate( PETSC_COMM_WORLD,PETSC_DECIDE,n,n,A,ierr )
call  MatSetFromOptions( A, ierr )
call  VecCreate( PETSC_COMM_WORLD,x,ierr )
call  VecSetSizes( x, PETSC_DECIDE, n, ierr )
call  VecSetFromOptions( x, ierr )
call  VecDuplicate( x,b,ierr )

C     then assemble matrix and right-hand-side vector

call  KSPCreate(PETSC_COMM_WORLD,ksp,ierr)
call  KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN,ierr)
call  KSPSetFromOptions(ksp,ierr)
call  KSPSolve(ksp,b,x,ierr)
call  KSPDestroy(ksp,ierr)

beginner
Customization Options

• **Command Line Interface**
  – Applies same rule to all queries via a database
  – Enables the user to have complete control at runtime, with no extra coding

• **Procedural Interface**
  – Provides a great deal of control on a usage-by-usage basis inside a single code
  – Gives full flexibility inside an application
Setting Solver Options at Runtime

- **-ksp_type** [cg, gmres, bcgs, tfqmr, …]
- **-pc_type** [lu, ilu, jacobi, sor, asm, …]

- **-ksp_max_it** <max_iters>
- **-ksp_gmres_restart** <restart>
- **-pc_asm_overlap** <overlap>
- **-pc_asm_type** [basic, restrict, interpolate, none]
- etc ...

solvers: linear
Linear Solvers: Monitoring Convergence

- **-ksp_monitor** - Monitor preconditioned residual norm
- **-ksp_monitor_solution** - Monitor solution graphically
- **-ksp_monitor_true_residual** - Monitor true residual norm $|| b-Ax ||$
- **-ksp_monitor_singular_value** - Monitor singular values

User-defined monitors, using callbacks
Setting Solver Options within Code

- `KSPSetType(KSP ksp, KSPType type)`
- `KSPSetTolerances(KSP ksp, PetscReal rtol, PetscReal atol, PetscReal dtol, int maxits)`
  - etc....
- • `KSPGetPC(KSP ksp, PC *pc)`
  - `PCSetType(PC pc, PCType)`
  - `PCASMSSetOverlap(PC pc, int overlap)`
  - etc....
Recursion: Specifying Solvers for Schwarz Preconditioner Blocks

• Specify KSP solvers and options with “-sub” prefix, e.g.,
  – Full or incomplete factorization
    - -sub_pc_type lu
    - -sub_pc_type ilu -sub_pc_ilu_levels <levels>
  – Can also use inner Krylov iterations, e.g.,
    - -sub_ksp_type gmres -sub_ksp_rtol <rtol>
    - -sub_ksp_max_it <maxit>
KSP: Review of Basic Usage

- KSPCreate( ) - Create solver context
- KSPSetOperators( ) - Set linear operators
- KSPSetFromOptions( ) - Set runtime solver options for [KSP,PC]
- KSPSolve( ) - Run linear solver
- KSPView( ) - View solver options actually used at runtime (alternative: -ksp_view)
- KSPDestroy( ) - Destroy solver
### KSP: Review of Selected Preconditioner Options

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set preconditioner type</td>
<td><code>PCSetType()</code></td>
<td><code>-pc_type [lu,ilu,jacobi, sor,asm,...]</code></td>
</tr>
<tr>
<td>Set level of fill for ILU</td>
<td><code>PCILUSetLevels()</code></td>
<td><code>-pc_ilu_levels &lt;levels&gt;</code></td>
</tr>
<tr>
<td>Set SOR iterations</td>
<td><code>PCSORSSetIterations()</code></td>
<td><code>-pc_sor_its &lt;its&gt;</code></td>
</tr>
<tr>
<td>Set SOR parameter</td>
<td><code>PCSORSSetOmega()</code></td>
<td><code>-pc_sor_omega &lt;omega&gt;</code></td>
</tr>
<tr>
<td>Set additive Schwarz variant</td>
<td><code>PCASMSSetType()</code></td>
<td><code>-pc_asm_type [basic, restrict,interpolate,none]</code></td>
</tr>
<tr>
<td>Set subdomain solver options</td>
<td><code>PCGetSubKSP()</code></td>
<td><code>-sub_pc_type &lt;pctype&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>-sub_ksp_type &lt;ksp_type&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>-sub_ksp_rtol &lt;rtol&gt;</code></td>
</tr>
</tbody>
</table>

1. beginner  
2. intermediate

And many more options...

solvers: linear: preconditioners
# Review of Selected Krylov Method Options

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Krylov method</td>
<td>KSPSetType( )</td>
<td>-ksp_type [cg,gmres,bcgs, tfqmr,cgs,...]</td>
</tr>
<tr>
<td>Set monitoring routine</td>
<td>KSPSetMonitor( )</td>
<td>-ksp_monitor,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-ksp_monitor_true_residual,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-ksp_singular_value</td>
</tr>
<tr>
<td>Set convergence tolerances</td>
<td>KSPSetTolerances( )</td>
<td>-ksp_rtol &lt;rt&gt; , -ksp_atol &lt;at&gt;</td>
</tr>
<tr>
<td>Set GMRES restart parameter</td>
<td>KSPGMRESRestart( )</td>
<td>-ksp_max_its &lt;its&gt;</td>
</tr>
<tr>
<td>Set orthogonalization routine for GMRES</td>
<td>KSPGMRESSetOrthogonalization( )</td>
<td>-ksp_gmres_restart &lt;restart&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-ksp_gmres_classicalgramschmidt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-ksp_gmres_modifiedgramschmidt</td>
</tr>
</tbody>
</table>

And many more options...

beginner  intermediate
Why Polymorphism?

• Programs become independent of the choice of algorithm
• Consider the question:
  – What is the best combination of iterative method and preconditioner for my problem?
• How can you answer this experimentally?
  – Old way:
  – New way:…
KSP: Runtime Script Example

```csh
#!/bin/csh

# Sample script: Experimenting with linear solver options.
# Can be used with, e.g., petsc/src/sles/examples/tutorials/ex2.c

foreach np (1 2 4 8)  # number of processors
    foreach ksptype (gmres bcgs tfqmr)  # Krylov solver
        foreach pctype (bjacobi asm)  # preconditioner
            foreach subpctype (jacobi sor ilu)  # subdomain solver
                if ($subpctype == ilu) then
                    foreach level (0 1 2)  # level of fill for ILU(k)
                        echo '***** Beginning new run *****'
                        mpirun -np $np ex2 -pc_type $pctype -ksp_type $ksptype \
                                -sub_ksp_type preonly sub_pc_type $subpctype \
                                -sub_pc_ilu_levels $level \
                                -ksp_monitor -sles_view -optionsleft
                else
                    echo '***** Beginning new run *****'
                    mpirun -np $np ex2 -pc_type $pctype -ksp_type $ksptype \
                            -sub_ksp_type preonly sub_pc_type $subpctype \
                            -ksp_monitor -sles_view -optionsleft
                endif
            end
        end
    end
end
```

solvers: linear

intermediate
Viewing KSP Runtime Options

舌样化 interventions

```bash
[lava] ex2 -ksp_monitor -pc_ilu_levels 1 -sles_view > out.5
0 KSP Residual norm 5.39418560416e+00
1 KSP Residual norm 1.238309089931e+00
2 KSP Residual norm 1.10413215450e-01
3 KSP Residual norm 6.60974009831e-03
4 KSP Residual norm 2.32911209560e-04
KSP Object:
    method: gmres
        GMRES: restart=30, using Modified Gram-Schmidt Orthogonalization
        maximum iterations=10000, initial guess is zero
        tolerances: relative=0.000138889, absolute=1e-50, divergence=10000
        left preconditioning
PC Object:
    method: ilu
        ILU: 1 level of fill
        out-of-place factorization
        matrix ordering: natural
        linear system matrix = precond matrix:
Matrix Object:
    type=MATSEQAIJ, rows=56, cols=56
    total: nonzeros=250, allocated nonzeros=560
    Norm of error 0.000280658 iterations 4
```

intermediate

solvers:
linear
Providing Different Matrices to Define Linear System and Preconditioner

Solve $Ax=b$

Precondition via: $M_L^{-1}AM_R^{-1}(M_Rx) = M_L^{-1}b$

• Krylov method: Use $A$ for matrix-vector products
• Build preconditioner using either
  – $A$ - matrix that defines linear system
  – or $P$ - a different matrix (cheaper to assemble)
• **KSPSetOperators**(KSP ksp,
  – Mat A,
  – Mat P,
  – MatStructure flag)
Matrix-Free Solvers

- Use “shell” matrix data structure
  - `MatCreateShell(…, Mat *mfctx)`

- Define operations for use by Krylov methods
  - `MatShellSetOperation(Mat mfctx,`  
    - `MatOperation MATOP_MULT,`  
    - `(void *) int (UserMult)(Mat, Vec, Vec))`  

- Names of matrix operations defined in
  `petsc/include/petscmat.h`

- Some defaults provided for nonlinear solver usage