Heat-Example with PETSc

Rolf Rabenseifner
rabenseifner@hlrs.de

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

Heat Example

- Compute steady temperature distribution for given temperatures on a boundary
- i.e., solve Laplace partial differential equation (PDE)

$$-\Delta u(x,y) = -[\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y)] = 0 \quad \text{on } \Omega \subset R^2$$

- with boundary condition

$$u(x,y) = \phi(x,y) \quad \text{on } \partial\Omega$$

- area $\Omega = [x_{min},x_{max}]\times[y_{min},y_{max}]$
- Compare:
  - Chap. [6] A Heat-Transfer Example with MPI
  - Explicit time-step integration of the the unsteady heat conduction

$$\partial u/\partial t = \Delta u$$

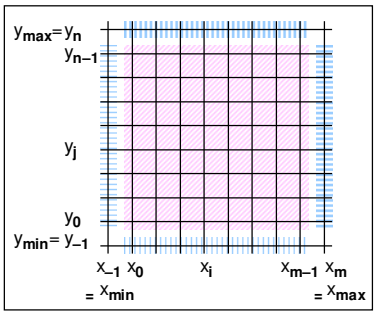
Discretization

- $x_i = (i+1)h + x_{min}$ $\quad i = -1, 0, \ldots, m-1, m$
- $y_i = (j+1)h + y_{min}$ $\quad j = -1, 0, \ldots, n-1, n$
- same discretization in x and y: $h = (x_{max} - x_{min})/(m+1)$
  $= (y_{max} - y_{min})/(n+1)$
- $u_{i,j} = u(x_i, y_j)$
- Boundaries ($\partial\Omega$)

$i = -1,$	$j = 0,\ldots,n-1$
$i = m,$	$j = 0,\ldots,n-1$
$i = 0,\ldots,m-1,$	$j = -1$
$i = 0,\ldots,m-1,$	$j = n$ *)

- Area to be solved ($\Omega - \partial\Omega$)

$i = 0,\ldots,m-1,$	$j = 0,\ldots,n-1$



*) corners are unused

From PDE to the linear difference equations system

- Differentiation:

$$\frac{\partial}{\partial x} u(x_{i+½}, y_j) = (u_{i+1,j} - u_{i,j}) / h$$

$$\frac{\partial^2}{\partial x^2} u_{i,j} = \left(\frac{\partial}{\partial x} u_{i+½,j} - \frac{\partial}{\partial x} u_{i-½,j}\right) / h$$

$$\frac{\partial^2}{\partial x^2} u_{i,j} = (u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) / h^2$$

$$\frac{\partial^2}{\partial y^2} u_{i,j} = (u_{i,j+1} - 2u_{i,j} + u_{i,j-1}) / h^2$$

- $-\Delta u(x,y) = 0$

$$\Rightarrow \boxed{-u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i,j+1} - u_{i+1,j} = 0 \qquad \text{for } i=0,\ldots,m-1,\ j=0,\ldots,n-1}$$

Boundary conditions in the linear difference equations system

- $\boxed{-u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i,j+1} - u_{i+1,j} = 0 \quad \text{for } i=0,\ldots,m-1,\ j=0,\ldots,n-1}$

⇒ **Boundary condition** are used in the equations with i=0, i=m-1, j=0, j=n-1:

i=0, j=0	➔			$+4u_{i,j}$	$-u_{i,j+1}$	$-u_{i+1,j}$	$= u_{-1,0}$	$+ u_{0,-1}$
i=0, 0<j<n–1	➔		$-u_{i,j-1}$	$+4u_{i,j}$	$-u_{i,j+1}$	$-u_{i+1,j}$	$= u_{-1,j}$	
i=0, j=n–1	➔		$-u_{i,j-1}$	$+4u_{i,j}$		$-u_{i+1,j}$	$= u_{-1,n-1}$	$+ u_{0,n}$
0<i<m–1, j=0	➔	$-u_{i-1,j}$		$+4u_{i,j}$	$-u_{i,j+1}$	$-u_{i+1,j}$	$=$	$u_{i,-1}$
0<i<m–1, 0<j<n–1	➔	$-u_{i-1,j}$	$-u_{i,j-1}$	$+4u_{i,j}$	$-u_{i,j+1}$	$-u_{i+1,j}$	$= 0$	
0<i<m–1, j=n–1	➔	$-u_{i-1,j}$	$-u_{i,j-1}$	$+4u_{i,j}$		$-u_{i+1,j}$	$=$	$u_{i,n}$
i=m–1, j=0	➔	$-u_{i-1,j}$		$+4u_{i,j}$	$-u_{i,j+1}$		$= u_{m,0}$	$+ u_{m-1,-1}$
i=m–1, 0<j<n–1	➔	$-u_{i-1,j}$	$-u_{i,j-1}$	$+4u_{i,j}$	$-u_{i,j+1}$		$= u_{m,j}$	
i=m–1, j=n–1	➔	$-u_{i-1,j}$	$-u_{i,j-1}$	$+4u_{i,j}$			$= u_{m,n-1}$	$+ u_{m-1,n}$

Matrix notation

- Ordering

  i,j= 0,0; 0,1; … 0,n-1; 1,0; 1,1; … 1,n-1; … m-1,0; … m-1,n-1

  ➔ I= 0; 1; … n-1; n; n+1; … 2n-1; … (m-1)n; … mn-1

- Matrix equation: Au=**b**

$$A = (A_{IJ})_{\substack{I=0,\,mn-1\\ J=0,\,mn-1}} = \begin{pmatrix} B & -\mathbf{I} & & & \\ -\mathbf{I} & B & -\mathbf{I} & & \\ & -\mathbf{I} & B & \ldots & \\ & & \ldots & \ldots & -\mathbf{I} \\ & & & -\mathbf{I} & B \end{pmatrix} \in R^{mn\times mn}$$

$$\text{with } B = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & -1 & 4 & \ldots & \\ & & \ldots & \ldots & -1 \\ & & & -1 & 4 \end{pmatrix} \in R^{n\times n}, \quad \mathbf{I} = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ldots & \\ & & & & 1 \end{pmatrix} \in R^{n\times n}$$

Matrix notation, continued

- $-\Delta u(x,y) = 0 \quad \Leftrightarrow \quad Au=\mathbf{b}$ with

$i=0,$	$j=0$	➔
$i=0,$	$0<j<n-1$	➔
$i=0,$	$j=n-1$	➔
$0<i<m-1,$	$j=0$	➔
$0<i<m-1,$	$0<j<n-1$	➔
$0<i<m-1,$	$j=n-1$	➔
$i=m-1,$	$j=0$	➔
$i=m-1,$	$0<j<n-1$	➔
$i=m-1,$	$j=n-1$	➔

$$u = \begin{pmatrix} u_{0,0} \\ \dots \\ u_{0,j} \\ \dots \\ u_{0,n-1} \\ \hdashline \dots \\ \hdashline u_{i,0} \\ \dots \\ u_{i,j} \\ \dots \\ u_{i,n-1} \\ \hdashline \dots \\ \hdashline u_{m-1,0} \\ \dots \\ u_{m-1,j} \\ \dots \\ u_{m-1,n-1} \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} u_{-1,0} + u_{0,-1} \\ \dots \\ u_{-1,j} \\ \dots \\ u_{-1,n-1} + u_{0,n} \\ \hdashline \dots \\ \hdashline u_{i,-1} \\ \dots \\ 0 \\ \dots \\ u_{i,n} \\ \hdashline \dots \\ \hdashline u_{m,0} + u_{m-1,-1} \\ \dots \\ u_{m,j} \\ \dots \\ u_{m,n-1} + u_{m-1,n} \end{pmatrix}$$

repeated for $0 < i < m-1$

Heat example: boundary and solution

- Boundary & Solution: $u(x,y) = x$ on $[0,1]\times[0,1]$
- $(u_{ij})_{i=-1,\dots,m,\ j=-1,\dots,n} =$

$$\begin{array}{c} \xrightarrow{\quad y \quad} \\ \downarrow x \end{array} \begin{pmatrix} u_{-1,-1} & u_{-1,0} & \dots & u_{-1,n-1} & u_{-1,n} \\ u_{0,-1} & u_{0,0} & \dots & u_{0,n-1} & u_{0,n} \\ \dots & \dots & \dots & \dots & \dots \\ u_{m-1,-1} & u_{m-1,0} & \dots & u_{m-1,n-1} & u_{m-1,n} \\ u_{m,-1} & u_{m,0} & \dots & u_{m,n-1} & u_{m,n} \end{pmatrix} := h \cdot \begin{pmatrix} 0 & 0 & \dots & 0 & 0 \\ 1 & 1 & \dots & 1 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ m & m & \dots & m & m \\ m+1 & m+1 & \dots & m+1 & m+1 \end{pmatrix}$$

with $h = \frac{1}{m+1} = \frac{1}{n+1}$

Example with n=m=4, general boundary

$$
\begin{pmatrix}
4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4
\end{pmatrix}
\cdot
\begin{pmatrix}
u_{0,0} \\ u_{0,1} \\ u_{0,2} \\ u_{0,3} \\ u_{1,0} \\ u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ u_{2,0} \\ u_{2,1} \\ u_{2,2} \\ u_{2,3} \\ u_{3,0} \\ u_{3,1} \\ u_{3,2} \\ u_{3,3}
\end{pmatrix}
=
\begin{pmatrix}
u_{-1,0} + u_{0,-1} \\ u_{-1,1} \\ u_{-1,2} \\ u_{-1,3} + u_{0,4} \\ u_{1,-1} \\ 0 \\ 0 \\ u_{1,4} \\ u_{2,-1} \\ 0 \\ 0 \\ u_{2,4} \\ u_{4,0} + u_{3,-1} \\ u_{4,1} \\ u_{4,2} \\ u_{4,3} + u_{3,4}
\end{pmatrix}
$$

Example with n=m=4, with solution & boundary u(x,y) := x

$$
\begin{pmatrix}
4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4
\end{pmatrix}
\cdot
\begin{pmatrix}
0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.6 \\ 0.6 \\ 0.6 \\ 0.8 \\ 0.8 \\ 0.8 \\ 0.8
\end{pmatrix}
=
\begin{pmatrix}
0.0 + 0.2 \\ 0.0 \\ 0.0 \\ 0.0 + 0.2 \\ 0.4 \\ 0 \\ 0 \\ 0.4 \\ 0.6 \\ 0 \\ 0 \\ 0.6 \\ 1.0 + 0.8 \\ 1.0 \\ 1.0 \\ 1.0 + 0.8
\end{pmatrix}
$$

Solving heat equation with PETSc *)

- Initialization of PETSc
- Initialization of matrix **A**
- Initialization of the boundary condition **b**
  - Data: Vector **u** := predefined exact solution
    Vector **b** := boundary condition (RHS)
    Vector **x** := approximate solution computed
  - Initialization of **b**, **u**: b, u – see previous slides [heat, u(*x*,*y*)=*x*]
    or with **–random_exact_sol**: u = random values, b := Au
- Solving **Ax**=**b**
- Checking the solution error_norm = $\| x - u \|_2$

*) based on **petsc/src/sles/examples/tutorials/ex2.c**

Initialization of PETSc

```
21: /* Include "petscsles.h" so that we can use SLES solvers. Note that this file
       automatically includes:
         petsc.h        - base PETSc routines    petscvec.h   - vectors
         petscsys.h     - system routines        petscmat.h   - matrices
         petscis.h      - index sets             petscksp.h   - Krylov subspace methods
         petscviewer.h  - viewers                petscpc.h    - preconditioners          */
28: #include petscsles.h
33: int main(int argc,char **args)
34: {
35:     Vec          x, b, u;     /* approx solution, RHS, exact solution */
36:     Mat          A;           /* linear system matrix */
37:     SLES         sles;        /* linear solver context */
38:     PetscRandom  rctx;        /* random number generator context */
39:     PetscReal    norm;        /* norm of solution error */
40:     int          i,j, I,J, Istart, Iend, ierr, m = 4, n = 4, its;
41:     PetscTruth   flg;
42:     PetscScalar  v, h, one = 1.0, neg_one = –1.0;
43:     KSP          ksp;         KSPType ksptype;  PC pc;   PCType pctype;
45:     PetscInitialize(&argc, &args, (char *)0, help);
46:     PetscOptionsGetInt(PETSC_NULL,"–m",&m,PETSC_NULL);
47:     PetscOptionsGetInt(PETSC_NULL,"–n", &n, PETSC_NULL);
```

Initialization of matrix A

```
55:  /*  When using MatCreate(), the matrix format can be specified at runtime.
     Also, the parallel partitioning of the matrix is determined by PETSc at runtime.
     Performance tuning note: For problems of substantial size, preallocation of matrix memory is crucial for
     attaining good performance. Since preallocation is not possible via the generic matrix creation routine
     MatCreate(), we recommend for practical problems instead to use the creation routine for a particular
     matrix format, e.g.,  MatCreateMPIAIJ() – parallel AIJ (compressed sparse row)
                           MatCreateMPIBAIJ() – parallel block AIJ
     See the matrix chapter of the users manual for details. */
69:  MatCreate(PETSC_COMM_WORLD,PETSC_DECIDE,PETSC_DECIDE,m*n,m*n,&A);
70:  MatSetFromOptions(A);
73:  /*  Currently, all PETSc parallel matrix formats are partitioned by contiguous chunks of rows
         across the processors. Determine which rows of the matrix are locally owned.  */
77:  MatGetOwnershipRange(A,&Istart,&Iend);
92:  for (I=Istart; I<Iend; I++) {
93:      v = –1.0;  i = I/n;  j = I – i*n;
94:      if (i>0)      {J = I – n;  MatSetValues(A, 1,&I, 1,&J, &v, INSERT_VALUES);}
95:      if (i<m–1) {J = I + n;  MatSetValues(A, 1,&I, 1,&J, &v, INSERT_VALUES);}
96:      if (j>0)      {J = I – 1;  MatSetValues(A, 1,&I, 1,&J, &v, INSERT_VALUES);}
97:      if (j<n–1)  {J = I + 1;  MatSetValues(A, 1,&I, 1,&J, &v, INSERT_VALUES);}
98:      v = 4.0;  MatSetValues(A,1,&I,1,&I,&v,INSERT_VALUES);
     }
107: MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
108: MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
```


Initialization of u, b, x

```
110: /*
     Create parallel vectors.
       – We form 1 vector from scratch and then duplicate as needed.
       – When using VecCreate(), VecSetSizes and VecSetFromOptions()
         in this example, we specify only the
         vector's global dimension; the parallel partitioning is determined at runtime.
       – When solving a linear system, the vectors and matrices MUST
         be partitioned accordingly. PETSc automatically generates
         appropriately partitioned matrices and vectors when MatCreate()
         and VecCreate() are used with the same communicator.
       – The user can alternatively specify the local vector and matrix
         dimensions when more sophisticated partitioning is needed
         (replacing the PETSC_DECIDE argument in the VecSetSizes() statement
         below).
     */
126: VecCreate(PETSC_COMM_WORLD,&u);
127: VecSetSizes(u,PETSC_DECIDE,m*n);
128: VecSetFromOptions(u);
129: VecDuplicate(u,&b);
130: VecDuplicate(b,&x);
```

Initializing the values of b (and u)

```
145:  PetscOptionsHasName(PETSC_NULL,"–random_exact_sol",&flg);
146:  if ( ! flg) {
          VecGetOwnershipRange(b,&Istart,&Iend);
          h = 1.0 / (m+1);
          for (I=Istart; I<Iend; I++) {
              v = 0;  i = I/n;  j = I – i*n;  h = 1/(m+1);
              if (i==0)      v = v +   /* u(-1,j): */  h * 0;
              if (i==m-1)  v = v +   /* u(m,j): */  h * (m+1);
              if (j==0)      v = v +   /* u(i,-1): */  h * (i+1);
              if (j==n-1)   v = v +   /* u(i, n): */  h * (i+1);
              if (v != 0) ; VecSetValues(b,1,&I,&v,INSERT_VALUES);
              v =  /* u(i, j): */  h * (i+1); VecSetValues(u,1,&I,&v,INSERT_VALUES);
          }
          VecAssemblyBegin(b); VecAssemblyEnd(b);
          VecAssemblyBegin(u); VecAssemblyEnd(u);
160:  } else {
          PetscRandomCreate(PETSC_COMM_WORLD,RANDOM_DEFAULT,&rctx);
          VecSetRandom(rctx,u);  PetscRandomDestroy(rctx);
          MatMult(A,u,b);
164:  }
167:  /* View the exact solution vector if desired */
169:  PetscOptionsHasName(PETSC_NULL,"–view_exact_sol",&flg);
170:  if (flg) {VecView(u,PETSC_VIEWER_STDOUT_WORLD);}
```


Solving Ax=b

```
173:  /* - - - - - - Create the linear solver and set various options - - - - - - - */
177:  /* Create linear solver context */
179:  SLESCreate(PETSC_COMM_WORLD,&sles);
182:  /* Set operators. Here the matrix that defines the linear system
          also serves as the preconditioning matrix. */
185:  SLESSetOperators(sles,A,A,DIFFERENT_NONZERO_PATTERN);
188:  /* Set linear solver defaults for this problem (optional).
          – By extracting the KSP (Krylov subspace methods) and PC (Preconditioner) contexts from
            the SLES context, we can then directly call any KSP and PC routines to set various options.
          – The following two statements are optional; all of these parameters could
            alternatively be specified at runtime via SLESSetFromOptions().
            All of these defaults can be overridden at runtime, as indicated below. */
198:  SLESGetKSP(sles,&ksp);
199:  KSPSetTolerances(ksp,1.e-2/((m+1)*(n+1)),1.e-50,PETSC_DEFAULT,PETSC_DEFAULT);
202:  /* Set runtime options, e.g., -ksp_type <type> -pc_type <type> -ksp_monitor -ksp_rtol <rtol>
          These options will override those specified above as long as SLESSetFromOptions()
          is called _after_ any other customization routines. */
208:  SLESSetFromOptions(sles);
211:  /* - - - - - - Solve the linear system - - - - - - - */
214:  SLESSolve(sles,b,x,&its);
```

Printing the solution

```
229:  /* Draw solution grid */

233:  PetscOptionsHasName(PETSC_NULL,"–view_sol_serial",&flg);
234:  if (flg) {VecView( x, PETSC_VIEWER_STDOUT_WORLD); }

236:  PetscOptionsHasName(PETSC_NULL,"–view_sol",&flg);
237:  if (flg) {
238:      PetscScalar  *xx;
239:      VecGetArray( x, &xx );
240:      VecGetOwnershipRange(x,&Istart,&Iend);
241:      PetscPrintf( PETSC_COMM_WORLD,
                       "Solution Grid (without boundary conditions):\n" );
242:      for (I=Istart; I<Iend; I++) {
243:        i = I/n;  j = I – i*n;
244:          PetscSynchronizedPrintf( PETSC_COMM_WORLD, "%8.6f  ", xx[I–Istart] );
245:          if (j == (n–1) ) PetscSynchronizedPrintf( PETSC_COMM_WORLD, "\n");
246:      }
247:      PetscSynchronizedFlush( PETSC_COMM_WORLD );
248:      VecRestoreArray( x, &xx );
249:  }
```


Printing the solution via X Window

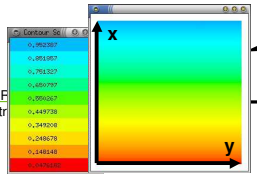
```
29: #include petscda.h

251:  PetscOptionsHasName(PETSC_NULL,"–view_sol_x",&flg);
252:  if (flg) { /* view solution grid in an X window */
253:      PetscScalar *xx;        DA  da;
          AO          ao;         Vec x_da;
257:      DACreate2d(PETSC_COMM_WORLD,DA_NONPERIODIC,DA_STENCIL_STAR,
258:        n,m,PETSC_DECIDE,PETSC_DECIDE,1,0,PETSC_NULL,PETSC_NULL,&da);
259:      DACreateGlobalVector(da, &x_da);
260:      DAGetAO(da, &ao);
261:      VecGetOwnershipRange(x, &Istart, &Iend);
262:      VecGetArray(x, &xx);
263:      for (I=Istart; I<Iend; I++)  {
264:        i = I; AOApplicationToPetsc(ao,1,&i);
265:        VecSetValues(x_da, 1, &i, &xx[I-Istart], INSERT_VALUES);
266:      }
267:      VecRestoreArray(x, &xx);
268:      VecAssemblyBegin(x_da); VecAssemblyEnd(x_da);
269:      PetscOptionsHasName(PETSC_NULL,"-view_sol_x_da",&flg);
270:      if (flg) VecView(x_da,PETSC_VIEWER_STDOUT_WORLD);
271:      VecView(x_da, PETSC_VIEWER_DRAW_(PETSC_COMM_WORLD));
272:      AODestroy(ao);
          DADestroy(da);
          VecDestroy(x_da);
273:  }
```

Create a 2-dimensional vector x_da

Copy values of x into x_da

Controling the distribution of x_da

View the solution as a 2-dimensional plot



Check solution and clean up

```
283:  /* Check the error */
285:  VecAXPY(&neg_one,u,x);
286:  VecNorm(x,NORM_2,&norm);
287:  /* Optional: Scale the norm:    norm *= sqrt(1.0/((m+1)*(n+1))); */

290:  /* Print convergence information. PetscPrintf() produces a single
         print statement from all processes that share a communicator.
         An alternative is PetscFPrintf(), which prints to a file.  */
294:  PetscPrintf(PETSC_COMM_WORLD,"Norm of error %A iterations %d\n",norm,its);

297:  /* Free work space.
          All PETSc objects should be destroyed when they are no longer needed. */
300:  SLESDestroy(sles);
301:  VecDestroy(u); VecDestroy(x);
302:  VecDestroy(b); MatDestroy(A);

305:  /* Always call PetscFinalize() before exiting a program. This routine
         – finalizes the PETSc libraries as well as MPI
         – provides summary and diagnostic information if certain runtime
           options are chosen (e.g., –log_summary). */
310:  PetscFinalize();
```


Program start

```
1: /* Program usage: mpirun –np <procs> ./heat_petsc [–help] [all PETSc options] */

3: static char help[ ] = "Solves a linear system in parallel with SLES: Compute steady   \n
4:                        temperature distribution for given temperatures on a boundary.\n
5:                        Input parameters include:                                     \n
6:                          –random_exact_sol : use a random exact solution vector      \n
7:                          –view_exact_sol   : write exact solution vector to stdout   \n
8:                          –view_sol_serial  : write solution grid to stdout (1 item/line) \n
9:                          –view_sol         : write solution grid to stdout (as matrix) \n
10:                         –view_sol_x –draw_pause 3 : view solution x on a X window   \n
11:                         –view_mat_x –draw_pause 3 : view matrix A on a X window     \n
12:                         –m <mesh_x>       : number of mesh points in x-direction    \n
13:                         –n <mesh_y>       : number of mesh points in y-direction    \n";
...
46:   PetscInitialize(&argc, &args, (char *)0, help);
```

Other Options

–help	prints **all** options
–ksp_type <type>	e.g., cg (**Conjugate Gradient**), cr (**Conjugate Residual**), bcgs (**BiCGSTAB**), cgs (**Conjugate Gradient Squared**), tfqmr (**Transpose-Free Quasi-Minimal Residual**), bicg (**BiConjugate Gradient**), qmres (**Generalized Minimal Residual**)
–ksp_rtol <rtol>	convergence criterion set by the program to **1.e-2/((m+1)*(n+1))**
–pc_type <type>	e.g., bjacobi (**BlockJacobi**), asm (**Additive Schwarz**)
–sub_pc_type <type>	e.g., jacobi (**Block Jacobi**), sor (**SOR**), ilu (**Incomplete LU**)
–ksp_monitor	prints an estimate of the l_2 -norm of the residual at each iteration
–sles_view	prints information on chosen KSP (solver) and PC (preconditioner)
–log_summary	prints statistical data
–options_table	prints all used options
–options_left	prints options table and unused options

Runtime Script Example, I.

```
1  #! /bin/csh
2  #
3  # Sample script: Experimenting with linear solver options.
4  # Can be used with, e.g., petsc/src/sles/examples/tutorials/ex2.c
5  # or heat_petsc.c
6  #
7  set appl='./heat_petsc'                          # path of binary
8  set options='–ksp_monitor –sles_view –log_summary –options_table –options_left
                –m 10 –n 10'
9  set num='0'
10 foreach np (1 2 4 8)                             # number of processors
11   foreach ksptype (gmres bcgs tfqmr)             # Krylov solver
12     set pctypes_parallel='bjacobi asm'           # parallel preconditioners
13     set pctypes_serial='ilu'                     # non-parallel preconditioners
14     if ($np == 1) then
15       set pctype_list="$pctypes_serial $pctypes_parallel"
16     else
17       set pctype_list="$pctypes_parallel"
18     endif
19     foreach pctype ($pctype_list)
20–49     ...          (see next slide)
50     end #for pctype
51   end #for ksptype
52 end #for np
```

Runtime Script Example, II.

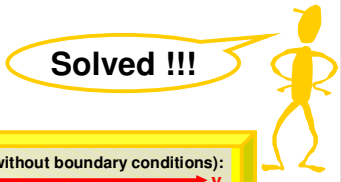
```
10  foreach  np (1  2  4  8)                    #  number of processors
11   foreach  ksptype (gmres  bcgs  tfqmr)      #  Krylov solver
12    set  pctypes_parallel='bjacobi  asm'      #  parallel preconditioners
13    set  pctypes_serial='ilu' #  non-parallel preconditioners
14    if ($np == 1) then ; set  pctype_list="$pctypes_serial  $pctypes_parallel"
16    else                ; set  pctype_list="$pctypes_parallel"
18    endif
19    foreach  pctype ($pctype_list)
20     if ($pctype == ilu) then                  #  non-parallel preconditioner
21      foreach  level (0  1  2)                 #  level of fill for ILU(k)
22        echo ' '
23        echo '************************ Beginning new run *************************'
24        echo ' '
25        set  cmd="mpirun  –np $np   $appl   –ksp_type  $ksptype   –pc_type  $pctype
                                –pc_ilu_levels  $level    $options"
26        set  num=`expr  $num + 1`;  echo "$num :  $cmd"
27        eval  $cmd
28       end #for level
29      else                                      #  parallel preconditioner
30–48     ...        (see next slide)
49      endif #pctype
50     end #for pctype
51    end #for ksptype
52  end #for np
```


Runtime Script Example, III.

```
20      if ($pctype == ilu) then                  #  non-parallel preconditioner
21–28 ...
29      else                                      #  parallel preconditioner
30       foreach  subpctype (jacobi  sor  ilu)    #  subdomain solver
31        if ($subpctype == ilu) then
32          foreach level (0 1 2)  #  level of fill for ILU(k)
34            echo '************************ Beginning new run *************************'
36            set  cmd="mpirun   –np $np   $appl   –ksp_type $ksptype   –pc_type  $pctype
                                –sub_ksp_type  preonly   –sub_pc_type  $subpctype
                                –sub_pc_ilu_levels  $level   $options"
37            set  num=`expr  $num + 1`;  echo "$num :  $cmd"
38            eval  $cmd
39          end #for level
40        else
42            echo '************************ Beginning new run *************************'
44            set  cmd="mpirun   –np $np   $appl   –ksp_type $ksptype   –pc_type  $pctype
                                –sub_ksp_type  preonly   –sub_pc_type  $subpctype
                                $options"
45            set  num=`expr  $num + 1`;  echo "$num :  $cmd"
46            eval  $cmd
47        endif #subpctype
48       end #for subpctype
49      endif #pctype
```

Output Example

```
t3e> setenv PETSC_DIR /usr/local/lib/PETSC ; setenv PETSC_ARCH t3e
t3e> make BOPT=O heat_petsc
t3e> mpirun –np 3 ./heat_petsc –ksp_type cg –m 4 –n 4 –ksp_monitor
                                  –sles_view –view_sol –log_summary –options_table –options_left
 0 KSP Residual norm 1.242025913946e+00
...
 6 KSP Residual norm 8.610435306905e–04
 7 KSP Residual norm 2.704366376622e–04
KSP Object:
 type: cg
 maximum iterations=10000, initial guess is zero
 tolerances: relative=0.0004, absolute=1e–50, divergence=10000
 left preconditioning
PC Object:
 type: bjacobi
   block Jacobi: number of blocks = 3
KSP Object:(sub_)
   type: preonly
   tolerances: relative=1e–05, absolute=1e–50,
   left preconditioning
 PC Object:(sub_)
   type: ilu
     ILU: 0 levels of fill
     ILU: max fill ratio allocated 1
     ILU: tolerance for zero pivot 1e–12
...
```



Solution Grid (without boundary conditions):

y →

0.199977	0.199891	0.199995	0.199985
0.400010	0.400072	0.400007	0.399864
0.600126	0.600078	0.599905	0.599989
0.800019	0.799936	0.799993	0.800014

x ↓

Norm of error 0.000269002 iterations 7

	Max	Max/Min	Avg	Total
Time (sec):	**7.033e–02**	1.01369	6.971e–02	
Objects:	4.100e+01	1.00000	4.100e+01	
Flops:	1.071e+03	1.28571	9.370e+02	**2.811e+03**
Flops/sec:	1.523e+04	1.26883	1.343e+04	**4.030e+04**

Makefile

```
ALL: heat_petsc

CFLAGS   =
FFLAGS   =
CPPFLAGS =
FPPFLAGS =

include ${PETSC_DIR}/bmake/common/base

heat_petsc: heat_petsc.o chkopts
<TAB> –${CLINKER} –o heat_petsc heat_petsc.o ${PETSC_SNES_LIB}
<TAB> ${RM} heat_petsc.o
```

Installation

- Set the environmental variable PETSC_DIR to the full path of the PETSc home directory, for example:
  setenv PETSC_DIR /home/username/petsc-2.1.3
- Set the environmental variable PETSC_ARCH, which indicates the architecture on which PETSc will be configured. For example, use
  setenv PETSC_ARCH solaris_gnu
  setenv PETSC_ARCH `$PETSC_DIR/bin/petscarch`
- In the PETSc home directory, type
  make BOPT=g all >& make_log
  to build a debugging version of the PETSc or
  make BOPT=O all >& make_log
  to build optimized version of the PETSc libraries.

Customized installation

Under the following circumstances it might be necessary to customize your installation of PETSc:

- packages like BLAS or Lapack are not installed in the default directories
- you want to use additional packages like Matlab or BlockSolve
- you want to use special compiler or linker options

Customized installation

The PETSc Makefile System is located in ${PETSC_DIR}/bmake.
This directory has subdirectories for each supported platform.
If you want to customize your installation you have
to edit the following files:

- ${PETSC_DIR}/bmake/${PETSC_ARCH}/packages
  - locations of all needed packages
- ${PETSC_DIR}/bmake/${PETSC_ARCH}/variables
  - definitions of compilers, linkers, etc.

Example - /bmake/linux/packages

```
# $Id: packages,v 1.63 2001/10/10 18:50:03 balay Exp $
# This file contains site-specific information.  The definitions below
# should be changed to match the locations of libraries at your site.
# The following naming convention is used:
#     XXX_LIB - location of library XXX
#     XXX_INCLUDE - directory for include files needed for library XXX
# Location of BLAS and LAPACK.
# See ${PETSC_DIR}/docs/intallation.html for information on
# retrieving them.
# BLASLAPACK_LIB      = -L/home/petsc/software/blaslapack/linux
                        -lflapack -lfblas
BLASLAPACK_LIB        = -L/home/petsc/software/mkl_linux/LIB
                        -lmkl32_lapack -lmkl32_def -lpthread
#
# Location of MPI (Message Passing Interface) software
#
MPI_HOME              = /home/petsc/software/mpich-1.2.0/linux
MPI_LIB               = -L${MPI_HOME}/lib -lmpich
MPI_INCLUDE           = -I${MPI_HOME}/include
MPIRUN                = ${MPI_HOME}/bin/mpirun -machinefile
                        ${PETSC_DIR}/maint/hosts.local
```

Example - /bmake/linux/packages

```
# ----------------------------------------------------------------------------
#  Locations of OPTIONAL packages. Comment out those
#  you do not have.
# ----------------------------------------------------------------------------
# Location of X-windows software
X11_INCLUDE          =
X11_LIB              = -L/usr/X11R6/lib -lX11
PETSC_HAVE_X11       = -DPETSC_HAVE_X11
# Location of MPE
# If using MPICH version 1.1.2 or higher use the flag
#DPETSC_HAVE_MPE_INITIALIZED_LOGGING
#MPE_INCLUDE         = -I/home/petsc/mpich-1.1.1/mpe
#MPE_LIB             = -L/home/petsc/mpich-1.1.1/lib/LINUX/ch_p4
                       -lmpe -lpmpich
#MPE_INCLUDE         =
#MPE_LIB             = -L${MPI_HOME}/lib -lmpe
#PETSC_HAVE_MPE      = -DPETSC_HAVE_MPE
```


Summary

- **Heat equation, PDE**: $-\Delta u(x,y) = 0$ on $\Omega \subset R^2$ with $\Omega = [x_{min},x_{max}]\times[y_{min},y_{max}]$
- **Boundary condition**: $u(x,y)$ given on $\partial\Omega$
- **Discretization**: $-u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i,j+1} - u_{i+1,j} = 0$ for $i=0\ldots m-1$, $j=0\ldots n-1$
- **4 Boundaries**: $i=-1$, $i=m$, $j=-1$, $j=m$
- **New ordering**: $(i,j)_{i=0..m-1,\ j=0..n-1}$ ➔ $I = 0..mn-1$
- **Matrix equation**: **Au = b**, **A**=sparse matrix, **b**=based on u on $\partial\Omega$, **u**=solution on $\Omega-\partial\Omega$
- **Example** with n=m=4, with solution & boundary **u(x,y) := x**
- **Linear Equation Solver (SLES) with PETSc**
  - MatSetValues(**A**, 1,&I, 1,&J, &v, INSERT_VALUES);
  - VecSetValues(**b**,1,&I,&v,INSERT_VALUES);
  - SLESSetOperators(sles,**A**,A,DIFFERENT_NONZERO_PATTERN);
  - SLESSolve(sles,**b**,**x**,&its); ➔ **x** is the solution vector, ordered with $I = 0..mn-1$
  - printing **x** in ordering $(i,j)_{i=0..m-1,\ j=0..n-1}$ (transposed)
- **mpirun** –np 3 ./heat_petsc –ksp_type cg –m 4 –n 4 –ksp_monitor –sles_view –view_sol –log_summary –options_table –options_left
- **Solved !!!**

Solution Grid (without boundary conditions):

y →

0.199977	0.199891	0.199995	0.199985
0.400010	0.400072	0.400007	0.399864
0.600126	0.600078	0.599905	0.599989
0.800019	0.799936	0.799993	0.800014

x

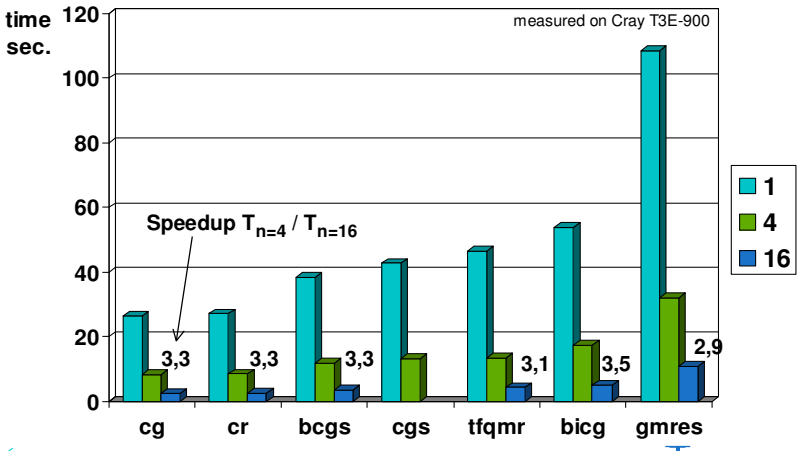
Norm of error 0.000269002 iterations 7

Practical

- **Test the heat_petsc example:**
  - cd ~/PETSC/#nr
  - mpirun –np 4 **./**heat_petsc
  - mpirun –np 4 **./**heat_petsc **–wrong_option –options_left**
  - mpirun –np 4 **./**heat_petsc **–ksp_monitor –view_mat_x –draw_pause 3** –op...
  - mpirun –np 4 **./**heat_petsc **–sles_view –view_sol –view_sol_x –draw_pause 3**
- **Which is default KSP? / Compare the execution time:**
  - mpirun –np 4 **./**heat_petsc **–m 300 –n 300** –log_summary –options_left
  - mpirun –np 4 **./**heat_petsc –m 300 –n 300 **–ksp_type cg** –log_summary –op...
  - mpirun –np 4 **./**heat_petsc –m 300 –n 300 **–ksp_type cr** –log_summary ...
  - mpirun –np 4 **./**heat_petsc –m 300 –n 300 **–ksp_type bcgs** –log_summary ...
- **Calculate Speeup of CG:**
  - mpirun **–np 1 ./**heat_petsc –m 300 –n 300 **–ksp_type cg** –log_summary ...
  - mpirun **–np 16 ./**heat_petsc –m 300 –n 300 **–ksp_type cg** –log_summary ...
- **If you want to compile:**
  - cp **../**source/heat_petsc.c **../**source/Makefile **./**
  - setenv PETSC_DIR ... or export PETSC_DIR=...
  - setenv PETSC_ARCH ... or export PETSC_ARCH=...
  - make BOPT=O heat_petsc

Results for heat_petsc (300x300) – time

for n = 1 PC is ilu
for n > 1 PC is bjacobi (sub=ilu)



Results for heat_petsc (300x300) – iterations

for n = 1 PC is ilu
for n > 1 PC is bjacobi (sub=ilu)