Parallel Programming Workshop

Numerical Libraries for Algebraic Problems

Hans-Peter Kersken

HLRS

January 29th,1998

HPF	implicit message passing all communication hidden from the user

$$\Downarrow$$

Parallel Library	communication takes place in library routines set-up of communication data structure hidden from the user some user controlled calls for efficiency

$$\Uparrow$$

MPI	explicit message passing all communication programmed by the user

Parallel Programs

- more complex
- more error prone
- more difficult to verify

Parallel Libraries

- allow high level view of data
- hide communication
- faster program development
- easy start-up

- Parallel Sparse Matrix Libraries
  - Overview
  - Petsc
    - Fundamentals
    - Examples
- Parallel Direct Solvers

SCALAPACK

- dense or band matrices
- BLACS
- PBLAS
- block cyclic data distribution

  !HPF$ DISTRIBUTE CYCLIC(MB), CYCLIC(NB)

- call interface similar to LAPACK
- linear equation solver, eigenproblem solver, orthogonal factorization
- http:/www.netlib.org/scalapack/index.html

Sparse Matrix Libraries

• PIM *(R. Da Cunha, T. D. Hopkins)*
www.arc.unm.edu/workshop/pmatlib/pimtut.html

Krylov subspace solvers

user supplied:
- preconditioner
- matrix vector multiply
- data layout
- data distribution

• BlockSolve95 *(M. T. Jones, P. E. Plasman)*
www.netlib.org/utk/cs_dept/research/jones/blocksolve/blocksolve.html

solver: CG, SYMMLQ, GMRES
preconditioner: ICC(0), ILU(0),
SOR(ω=1), block Jacobi

user supplied:
- elementary matrix and vector operation
- data layout
- data distribution

- Aztec *(S. A. Hutchinson, J. N. Shadid, R. S. Tuminaro)*
  www.cs.sandia.gov/HPCCIT/aztec.html

solver: CG, GMRES, CGS, TFQMR, Bi-CGSTAB
preconditioner: Jacobi, block Jacobi
elementary matrix and vector operation

user supplied:
- data layout
- data distribution

- PSPARSLIB (Y. SAAD)
  http://www.cs.umn.edu/~saad/software.html

solver: CG, GMRES, Bi-CGSTAB, QMR,
preconditioner: Schwarz-type, ILU, multicolor SOR
elementary matrix and vector operation

user supplied:
- data layout
- data distribution

PETSc

Portable Extensible Toolkit for Scientific Computation

S. Balay, W. Gropp, L. C. McInnes, B. Smith

Argonne National Laboratory

http://www.mcs.anl.gov/petsc

- elementary matrix and vector operation
- real and complex arithmetics
- data distribution mechanism
- iterative linear solvers :

  Richardson, Chebyshev, Jacobi, SOR,
  CG, GMRES, Bi-CGSTAB, CGS, TFQMR

- preconditioners:
    - sequential: ILU, ICC, SSOR, block GS, LU
    - parallel : Jacobi, block Jacobi, additive Schwarz
    - interface to parallel ICC and ILU from BlockSolve95
- non linear solvers
- support for time-dependent problems
- error checking
- profiling
- user supplied:

    data layout

- platforms:

IBM-SP2	Cray-T3D
SGI	Cray-T3E
SUN	Dec-Alpha
Intel Paragon	HP
Linux PC	NT PC

- extensive documentation

- support

- only elementary knowledge of MPI required

PETSC DESIGN

- ANSI C, FORTRAN77, Fortran90

- Data encapsulation
  VecSetValue(), VecGetArray()

- Polymorphism
  e. g. VecDot(Vec x, Vec y, Scalar r) regardless of internal vector data structure

- Context variables
  SlesCreate(&sles, ...)

Performance

- overlapping communication and computation
- precompute and optimize communication structure
- aggregate data for communication
- user driven communication events

Ease of use

- parallel data objects
- communication hiding

PETSC OBJECTS

- Data objects - encapsulate data
    - vectors
    - matrices
    - distributed arrays
    - index set
- Context objects - encapsulate state and control information
    - linear solver
    - Krylov subspace methods
    - preconditioners
    - graphic

S

VECTORS

Initialize

```
Vec x, y, z ;
int ierr;
int local_length, global_length, n_vecs;
Vec new_vec, *new_vecs;
Scalar *vals = (Scalar *) PetscMalloc(sizeof(Scalar) * local_length);

ierr = VecCreateSeq(MPI_COMM_SELF, local_length, &x);

ierr = VecCreateMPI(MPI_COMM_WORLD, local_length, global_length, &x);

ierr = VecCreateMPI(MPI_COMM_WORLD, PETSC_DECIDE, global_length, &y);

ierr = VecCreateMPI(MPI_COMM_WORLD, local_length, PETSC_DECIDE, &z);

ierr= VecCreateMPIWithArray(MPI_COMM_WORLD, local_length,global_length, vals, &x)

ierr = VecDuplicateVec(x, &new_vec);

ierr = VecDuplicateVecs(n_vecs, x, &new_vecs);
```

Set values

```
Scalar a;
Vec x;
int n_indices;
int *indices;
int indices[];
Scalar values[];

ierr = VecSet(&a, x);

ierr = VecSetValue(x, index, a, INSERT_VALUES);

ierr = VecSetValues(x, n_indices, indices, values, INSERT_VALUES);
```

Assembling

```
VecAssemblyBegin(x);
VecAssemblyEnd(x);
```

AccessValues

```
Vec x;
Scalar *array;
int length;


ierr = VecGetArray(x, &array);
•
•
•
•
ierr = VecRestoreArray(x, &array);
```

Basic Vector Operation

Vec x, y, w;
Scalar r;
Scalar a;
Norm_Type type;
int n;

VecAXPY(&a, x, y)	y=y+a*x
VecAYPX(&a, x, y)	y=x+a*y
VecWAXPY(&a, x, y, w)	w=a*x+y
VecAXPBY(&a, &, x, y)	y=a*x+b*y
VecScale(&a, x)	x=a*x
VecDot(x, y, &r)	r=x'*y
VecTDot(x, y, &r)	r=x'*y
VecNorm(x, type, &r)	r=||x||_type
VecSum(x, &r)	r=x_i
VecCopy(x, y)	y=x
VecSwap(x, y)	y=t; y=x; x=t
VecPointwiseMult(x, y, w)	w_i=x_i*y_i
VecPointwiseDivide(x, y, w)	w_i=x_i/y_i
VecMDot(n, x, &y, &r)	r(i)=x'*y(i)
VecMTDot(n, x, &y, &r)	r(i)=x'*y(i)
VecMAXPY(n, &a, x, &y)	y(i)=a_i*x+y(i)
VecMax(x, &idx, &r)	r=max(x_i)
VecMin(x, &idx, &r)	r = min(x_i)
VecReciprocal(x)	x_i = 1/x_i
VecShift(&s, x)	x_i = s + x_i
VecAbs(x)	x_i = |x_i|

MATRICES

Initialization

```
MPI_Comm comm;
int M, N;
int m, n;
Mat A;
int d_nz;
int d_nnz[];
int o_nz;
int o_nnz[];

ierr = MatCreate( comm, M, N, &A);

ierr = MatCreateMPIAIJ(comm, m, n, M, N, 0, PETSC_NULL, 0, PETSC_NULL, &A);

ierr = MatCreateMPIAIJ(comm, m, n, M, N, d_nz, d_nnz, o_nz, o_nnz, &A);
```

Set values

```
Mat A
int n_rows, n_cols;          /* size of block to be set */
Scalar *values;              /*  value of entry */
Scalar value;                /*  values of entries */
int i_row, i_col;            /* global row and column index to be inserted */
int *i_row, *i_cols;         /* list of global row and column indices to be inserted */

/* set a single value */

ierr = MatSetValues(A, 1, &i_row, 1, &i_col, &value, INSERT_VALUES);

/* set a single row */

ierr = MatSetValues(A, 1, &i_row, n_cols, i_cols, values, INSERT_VALUES);

/* set a block */

ierr = MatSetValues(A, n_rows, i_rows, n_cols,  i_cols, values, INSERT_VALUES);
```

Assembling

```
ierr = MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);

ierr = MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);
```

Access to matrix entries

```
Mat A ;
int row;
int n_cols;
int *cols;
Scalar *vals;

ierr = MatGetRow(A, row, &ncols, &cols, &vals);
```

Basic operation

Scalar a;
Mat A, B;
MatType type;
NormType norm;
Scalar r;
Vec x;

Function	Operation
MatAXPY(&a, A, B);	Y=Y+a*X
MatMult(A, x, y)	y=A*x
MatMultAdd(A, x, y, z)	z=y+A*x
MatMultTrans(A, x, y)	$y=A^T*x$
MatMultTransAdd(A, x, y, z)	$z=y+A^T*x$
MatNorm(A, norm, &r)	r=||A||_norm
MatDiagonalScale(A, l, r)	A=diag(l)*A*diag(r)
MatScale(&a,A)	A=a*A
MatConvert(A, type, &B)	B=A
MatCopy(A,B)	B=A
MatGetDiagonal(A, x)	x=diag(A)
MatTranspose(A, &B)	$B=A^T$
MatZeroEntries(A)	A=0

Matrix-Vector Multiplication

Simple Algorithm

- duplicate vector on each processor
- perform mutiplication

Efficient Algorithm

- split local rows into diagonal and off diagonal part
- set-up data structure for communication
- initiate non blocking communication
- perform multiplication of local part
- receive non local vector entries
- perform remaining operation

$$
\left(\begin{array}{ccc|ccc|ccc}
1 & 2 & 3 & 4 & & & & & \\
6 & 7 & 8 & 9 & & & & & \\
 & 4 & 4 & 4 & 6 & & & & \\
\hline
 & & 7 & 7 & 0 & 7 & & & \\
 & & & 8 & 8 & 8 & & & \\
 & 4 & & & 2 & 3 & 4 & & \\
\hline
 & 4 & 5 & & 5 & 6 & 6 & 1 & 1 \\
 & & & 1 & 4 & & 3 & 7 & 7 \\
 & & & & & & 5 & 9 & 9
\end{array}\right)
\begin{pmatrix} 1 \\ 2 \\ \underline{3} \\ 4 \\ 5 \\ \underline{6} \\ 7 \\ 8 \\ 9 \end{pmatrix}
$$

Matrix-Vector Example

$$\begin{pmatrix} 2 & -1 & & & & & & \\ -1 & 2 & -1 & & & \mathbf{0} & & \\ & -1 & 2 & -1 & & & & \\ & & -1 & 2 & -1 & & & \\ & & & -1 & 2 & -1 & & \\ & & & & -1 & 2 & -1 & \\ & \mathbf{0} & & & & -1 & 2 & -1 \\ & & & & & & -1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

```
#include "petsc.h"
#include "mat.h"
#include "vec.h"

int main(int argc,char **args)
{
  Vec      x, y;
  Mat      A;
  Scalar    norm;
  int       n=5;
  int flg;
  int Istart, Iend;
  Scalar v;
  int ierr;
  int I,J;
  Scalar one=1.;
  char *help="matrix vector multiply";

 ierr = PetscInitialize(&argc, &args, PETSC_NULL, help); CHKERRA(ierr);
```

```
/* get size of matrix from command line options */

  ierr = OptionsGetInt(PETSC_NULL,"-n",&n,&flg); CHKERRA(ierr);

/* simple matrix creation */

  ierr = MatCreate(MPI_COMM_WORLD, n, n, &A); CHKERRA(ierr);

/* get local range of rows */

  ierr = MatGetOwnershipRange(A, &Istart, &Iend); CHKERRA(ierr);

/* set diagonal and first sub and super diagonal */

v = 2.;
for ( I=Istart; I<Iend; I++ ) {
      J = I;
      ierr = MatSetValues(A, 1, &I, 1, &J, &v, INSERT_VALUES); CHKERRA(ierr);
}
```

```
v = -1;
for ( I=Istart; I<Iend; I++ ) {
      J = I - 1;
      if (I > 0) ierr = MatSetValues(A, 1,&I, 1, &J, &v, INSERT_VALUES); CHKERRA(ierr);
}

v = -1;
for ( I=Istart; I<Iend; I++ ) {
      J = I + 1;
      if (I < n-1) ierr = MatSetValues(A, 1, &I, 1, &J, &v, INSERT_VALUES); CHKERRA(ierr);
}

ierr = MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY); CHKERRA(ierr);
ierr = MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY); CHKERRA(ierr);

/* setup vectors */

ierr = VecCreate(PETSC_COMM_WORLD, n, &x); CHKERRA(ierr);

ierr = VecSet( &one, x); CHKERRA(ierr);
ierr = VecAssemblyBegin(x); CHKERRA(ierr);
ierr = VecAssemblyEnd(x); CHKERRA(ierr);
```

```
/* create new vector similar to x */

ierr = VecDuplicate(x, &y); CHKERRA(ierr);

/* finally multiply A and x , store result in y */

ierr = MatMult(A, x, y); CHKERRA(ierr);

ierr =  VecView(y, VIEWER_STDOUT_WORLD); CHKERRA(ierr);

ierr = PetscFinalize();CHKERRA(ierr);
}
```

```
> mat_x_vec -n 4
1
0
0
1
>mpirun -np 4  mat_x_vec -n 10
Processor (0)
1
0
0
Processor (1)
0
0
0
Processor (2)
0
0
Processor (3)
0
1
```

LINEAR SOLVERS

Simple Linear Equation Solver - SLES

- iterative linear solvers :

  Richardson, Jacobi, SOR, Chebyshev, CG, Bi-CGSTAB, CGS, TFQMR, GMRES

- preconditioners:
  - sequential: ILU, ICC, SSOR, block GS, LU
  - parallel : Jacobi, block Jacobi, additive Schwarz
  - interface to parallel ICC and ILU from BlockSolve95
  - suite of routines for implementing multigrid preconditoners
  - user defined
  - parpre-Interface

Interface

```
/* solve Ax=b */

MPI_Comm comm;
SLES sles;
Mat A, P;
MatStructure flag=DIFFERENT_NON_ZERO_PATTERN;
Vec, b, x;
int n_its;

ierr = SLESCreate(comm, &sles);

ierr = SLESSetOperators(sles, A, P, flag);

ierr = SLESSetFromOptions(sles);

ierr = SLESSolve(sles, b, x, &n_its);
```

```
MPI_Comm comm;
SLES sles;
KSP ksp;
PC pc;
Mat A, P;
MatStructure flag=DIFFERENT_NON_ZERO_PATTERN;
Vec, b, x;
int n_its, maxits;
double rtol, atol, dtol;

ierr = SLESCreate(comm, &sles);

ierr = SLESSetOperators(sles, A, P, flag);

ierr = SLESGetKSP(sles, &ksp);  ierr = KSPSetType(ksp, KSPCG);
ierr = KSPSetTolerances(ksp, rtol, atol, dtol, maxits)

ierr = SLESGetPC(sles, &pc);  ierr = PCSetType(pc, PCJACOBI);

ierr = SLESSetFromOptions(sles);
ierr = SLESSolve(sles, b, x, &n_its);
```

```
#include "petsc.h"
#include "sles.h"
char *help="simple SLES example. Poisson equation in a square";

int main(int argc, char **args)
{
    Vec     x, b, tmp;
    Mat A;
    Scalar  norm;
    int     n_x=5;
    int     n=n_x*n_x;
    int     flg;
    int     Istart, Iend;
    Scalar  v;
    int     ierr;
    int     I,J;
    int     i,j;
    Scalar  one=1., none = -1.;
    SLES    sles;
    KSP     ksp;
    PC      pc;
    int     n_its;
```

```
MatStructure flag =DIFFERENT_NONZERO_PATTERN;


ierr =  PetscInitialize(&argc, &args, PETSC_NULL, help); CHKERRA(ierr);

/* get grid size  from command line options */

ierr = OptionsGetInt(PETSC_NULL, "-nx", &n_x, &flg); CHKERRA(ierr);
if (flg != 0) n= n_x * n_x;

/* simple matrix creation */

ierr = MatCreate(PETSC_COMM_WORLD, n, n,&A); CHKERRA(ierr);

/* get local range of rows */

ierr = MatGetOwnershipRange(A,&Istart,&Iend); CHKERRA(ierr);
```

```
/* Set matrix elements for the 2-D, five-point stencil in parallel.
   assume zero boundary condition. 1D partition*/

for(I=Istart;I<Iend;I++){
    v  = -1.0;
    i  =  I / n_x;
    j = I - i * n_x;
    if(i>0){
        J = I-n_x;
        ierr = MatSetValues(A, 1, &I, 1, &J, &v, INSERT_VALUES); CHKERRA(ierr); }
    if(i<n_x-1){
        J = I+n_x;
        ierr = MatSetValues(A, 1, &I, 1, &J, &v, INSERT_VALUES); CHKERRA(ierr); }
    if(j>0){
        J = I-1;
        ierr = MatSetValues(A, 1, &I, 1, &J, &v, INSERT_VALUES); CHKERRA(ierr); }
    if(j<n_x-1){
        J = I+1;
        ierr = MatSetValues(A, 1, &I, 1, &J, &v, INSERT_VALUES); CHKERRA(ierr); }
    v = 4.0;
    ierr = MatSetValues(A, 1, &I, 1, &I, &v, INSERT_VALUES);
}
```

```
ierr = MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY); CHKERRA(ierr);
ierr = MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY); CHKERRA(ierr);

/* setup vectors */

ierr = VecCreate(PETSC_COMM_WORLD, n, &b); CHKERRA(ierr);

ierr = VecSet( &one, b); CHKERRA(ierr);
ierr = VecAssemblyBegin(b); CHKERRA(ierr);
ierr = VecAssemblyEnd(b); CHKERRA(ierr);

/* create new vectors similar to b */

ierr = VecDuplicate(b, &x); CHKERRA(ierr);
ierr = VecDuplicate(b, &tmp); CHKERRA(ierr);

/* Setup SLES */

ierr = SLESCreate(MPI_COMM_WORLD, &sles); CHKERRA(ierr);

ierr = SLESSetOperators(sles, A, A, flag); CHKERRA(ierr);
```

```
ierr = SLESGetKSP(sles, &ksp);CHKERRA(ierr);

ierr = KSPSetType(ksp, KSPCG);CHKERRA(ierr);

ierr = SLESGetPC(sles, &pc);CHKERRA(ierr);
ierr = PCSetType(pc, PCJACOBI);CHKERRA(ierr);

ierr = SLESSetFromOptions(sles); CHKERRA(ierr);

ierr = SLESSolve(sles, b, x, &n_its); CHKERRA(ierr);
PetscPrintf(MPI_COMM_WORLD,"number of iterations = %d\n", n_its);

/* calculate residual */

ierr = MatMult(A,x,tmp); CHKERRA(ierr);
ierr = VecAXPY(&none, b, tmp); CHKERRA(ierr);
ierr = VecNorm(tmp, NORM_2, &norm); CHKERRA(ierr);

PetscPrintf(MPI_COMM_WORLD,"residual norm = %e\n", norm);

ierr = PetscFinalize();CHKERRA(ierr);}
```