

7 Linear system solving on DMM

several parallel softwares for solving EDP have a linear algebra kernel

numerous softwares or libraries for linear systems or eigenproblems

list of free softwares

http://www.netlib.org/utk/people/JackDongarra/la-sw.html

One looks for a library using MPI

- ScaLAPACK

direct methods for dense and banded matrices, eigenvalues and eigenvectors

~same algorithms as LAPACK

cyclic distribution of matrix blocks

only Fortran

- PLAPACK

direct methods only for dense matrices

"natural" distribution of matrix blocks

C, Fortran, Matlab interfaces

PETSC (Portable Extensible Toolkit for Scientific Computations)

iterative methods for sparse matrices, preconditioning

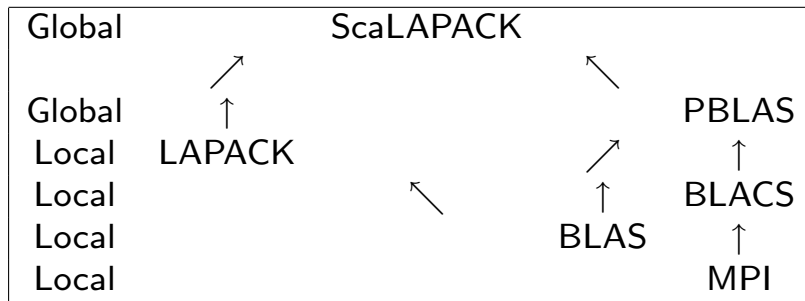
- SLEPc: eigenvalues, eigenvectors

C/C++, Fortran interfaces

7.1 Dense, banded matrices: Scalapack

Advantages: continuation of the Lapack project, same functions

Software structure



SPMD model

Single Program Multiple Data

—→ the same code is executed on different processors or processes and operate on different data. The difference between the instructions (*MIMD*) is obtained by checking the processor identity

7.1.1 Communication: BLACS

Goal: matrices (and vectors) transfer between different processors

Documentation on Moodle

Owns (and uses) some MPI features

Array communication
Process in 2D a cartesian grid

Example: 8 processes in a 2×4 grid

	0	1	2	3
0	0	1	2	3
1	4	5	6	7

- process identified by its cartesian coordinates
- an operation involving more than one sending process and more than one receiving process is called **scoped operation** and the processes in this operation type belong to the **range** of the operation
- Notion of context:
 - analogous to the MPI communicator
 - each grid is internally logically defined by a context
 - for the user, it is simply an integer
- most of the functions are of type *void*

Function names:

Fortran	C
BLACS_XXXXX	Cblacs_XXXXX
XXXXX	CXXXXX

Utilities summary

1. Initialization

Cblacs_pinfo: number of available processes

Cblacs_get: gives the value of the default context

Cblacs_gridinit: defines a grid, initializes its context

2. Destruction

Cblacs_exit: terminates the use of BLACS

Cblacs_gridexit: releases a grid

3. Information

Cblacs_gridinfo: properties of a grid in a given context

int Cblacs_pnum: rank of a processor of given coordinates

Cblacs_pcoord: coordinates of a processor of given rank

4. Synchronization

Cblacs_barrier: set a barrier (either on the whole grid , or on the rows or columns)

Example: grid 2×4

```
#include <stdio.h>

#include "mpi.h"

main(int argc, char* argv[])

{ int myid; /* process rank */

  int p; /* number of processes */

  int ctxt, pr=2, pc=4, myrow, mycol; /* BLACS stuff*/

  /* Start the MPI engine */

  MPI_Init(&argc, &argv);

  /* Find out number of processes */

  MPI_Comm_size(MPI_COMM_WORLD, &p);

  if (p<pr*pc)

    { printf("\n not enough processors \n");
```

```

    exit(0); }

/* Find out process rank */

    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

/* Get a BLACS context */

    Cblacs_get(0, 0, &ctxt);

/* Initialize the pr x pc process grid */

    Cblacs_gridinit(&ctxt, "Row-major", pr, pc);

    Cblacs_pcoord(ctxt, myid, &myrow, &mycol);

    :

/* Release process grid */

    Cblacs_gridexit(ctxt);

/* Shut down MPI */

    MPI_Finalize();

}

```

Communications summary

1. Point to point

Cdgesd2d: the current process sends a general $m \times n$ double precision matrix to the specified receiver

Cdgerv2d: the current process receives a matrix sent by a specified process

One sends an array to the buffer and one receives from the buffer into an array

2. Collective

Cdgebs2d: diffusion of a general matrix to all processes in the specified range

Cdgebr2d: receipt of a diffused matrix

3. Reduction

Cdgsum2d: entry by entry sum of a matrix of processes in the given range

Cdgamn2d: idem for the minimum

Cdgamx2d: idem for the maximum

7.1.2 Parallel basic functions: PBLAS

Remark: PBLAS and ScaLAPACK in the same library

PBLAS uses BLACS for communications

⇒ Initialize and finalize the use of BLACS (and of MPI too)

PBLAS has the same functions as BLAS

Example: DGEMV

BLAS: $y \leftarrow \alpha Ax + \beta y$ (trans='N')

```
void dgemv_(char* trans, int* m, int* n, double* alpha,
```

```
double* a, int* lda, double* x, int* incx,
```

```
double* beta, double* y, int* incy);
```

PBLAS: $sub(y) \leftarrow \alpha sub(A)sub(x) + \beta sub(y)$ (trans='N')

$sub(A) := A_{ia+i, ja+j}$, $0 \leq i \leq m-1$, $0 \leq j \leq n-1$

$sub(x) = x_{ix, jx+j}$, $0 \leq j \leq n-1$ if $incx = 1$, idem for $sub(y)$

```
pdgemv_(char* trans, int* m, int* n, double* alpha,
```

```
double* a, int* ia, int* ja, int* desca, double* x,
```

```
int* ix, int* jx, int* descx, int* incx, double* beta,
```

```
double* y, int* iy, int* jy, int* descy, int* incy)
```

dgemv_ works on arrays a,b,x

pdgemv_ works on subarrays sub(A), sub(x)... of global arrays distributed among different processes

the descriptors desca, descx and descy describe the distribution of the arrays A, x and y

All the processes call pdgemv_ and use the local distribution of the arrays A, x et y

Descriptor

```
int desc[9]
```

desc[0]=d_type_A	matrix; 1: dense
desc[1]=icontxt	BLACS context
desc[2]=m_A	nb of rows of A
desc[3]=n_A	nb of columns of A
desc[4]=mb_A	nb of rows per block
desc[5]=nb_A	nb of columns per block
desc[6]=rsrc_A	coord. of proc. $\supset A[0][.]$
desc[7]=csrc_A	coord. of proc. $\supset A[.][0]$
desc[8]=lld	nb of rows of the local arr. a

The variables $desc[j]$, $j = 0, 1, \dots, 7$ are global, $desc[8]$ is local to each process.

Initialization of the descriptor \implies

```
int desc_A[9];
```

```
desc_init_(desc_A, m, n, mb, nb, rsrc, csrc, ictxt, lld, info);
```

info=0 if OK, =-i if the i-th parameter has an illicit value

Typical program (PBLAS or ScaLAPACK)

One does $y = A * x$ with *pdgemv*

```
#include <stdio.h>
```

```
#include "mpi.h"
```

```
int m=16,n=20, mb=4,nb=4, mA=8,nA=12, zero=0,one=1;
```

```
int main(int argc, char* argv[])
```

```
{ int ctxt, nprow=2, npcol=2, myrow, mycol; /* BLACS stuff*/
```

```
int descA[9], descx[9], descy[9], info; /* BLACS stuff*/
```

```
double A[nA][mA], x[nA], y[mA]; /* local size of the arrays */
```

```
double alpha=1, beta=0,aux[nb*mb];
```

```
int i,j,ib,jb,iad,jad,nout=6;
```

```
/* Initialize mpi, blacs and the process grid */
```

```
MPI_Init(&argc, &argv);
```

```
Cblacs_pinfo(&iam,&nprocs);
```

```
if (iam==0 && nprocs<4)
```

```
{ printf("\n Number of processors should be at least 4\n");
```

```
return 0; }
```

```

Cblacs_get(0,0,&ctxt);

Cblacs_gridinit(&ctxt," Row-major" ,nprow,npcol);

/* Find who I am */

Cblacs_gridinfo(ictxt,&nprow,&npcol,&myrow,&mycol);

/* Initialize BLACS descriptors */

descinit_(descA,&m,&n,&mb,&nb,&zero,&zero,&ctxt,&m,&info);

descinit_(descx,&one,&n,&one,&nb,&zero,&zero,&ctxt,&one,&info);

descinit_(descy,&m,&one,&mb,&one,&zero,&zero,&ctxt,&m, &info);

/* Initialize matrix A with A[i,j] = i-j */

:

/* Initialize vector x = [0,1,...,N-1] */

:

/* Multiply y = alpha*A*x + beta*y (alpha=1, beta=0) */

pdgemv_(&nt,&m,&n,&alpha,&A[0][0],&one,&one,descA,

x,&one,&one,descx,&one,&beta,y,&one,&one,descy,&one);

```

```

/* Print result */

pdlaprnt_(&m,&n,y,&one,&one,&descy,&myrow,&mycol," y" ,&nout,aux)

/* Release process grid, shut down MPI */

Cblacs_gridexit(ctxt);

MPI_Finalize();

return 0;

}

```

Distribution of vectors and matrices

- It is the main difficulty of PBLAS and ScaLAPACK
- distribute a matrix from the central memory of the master processor or from a disk to all the processors
- better: each processor computes the part of the matrix that its memory shall contain!

Structure with three levels

1. Matrix A $m \times n$ (vector: matrix $m \times 1$ or $1 \times n$)
2. A partitioned by blocks of mb rows and nb columns
3. blocks distributed on the p processes of the $n_{prow} \times n_{pcol}$ grid

Receipe: blocks of size ≈ 64 , $p = mn/10^6$.

Block cyclic distribution

Algorithm: A partitioned blockwise

1. distribute the blocks of the 1st row to the 1st row of processes; if not enough rows of processes, restart at the 1st column
2. same operation for distributing the 2nd row of blocks to the 2nd row of processes
3. repeat the operation until the last row of processes
4. restart distributing the next row of blocks to the 1st row of processes

The blocks of a vector shall be cyclicly distributed on the processes of the 1st row or the first column of the grid

Example: Matrix partitioned in 4×5 blocks, grid of 2×2 processors

Blocks are numbered by rows

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

and distributed to the grid

1	3	5	2	4
11	13	15	12	14
6	8	10	7	9
16	18	20	17	19

7.1.3 ScaLAPACK

ScaLAPACK: Scalable Linear Algebra PACKage

\implies performance increases when the problem size or the number of processors increase

The block cyclic distribution allows to minimize the data transfer between the processes during the linear algebra operations

The features of ScaLAPACK are almost the same as those of LAPACK:

- Matrices: dense or dense banded
 $d_type = 1$: dense, $d_type = 501$ banded or tridiagonal
- Solution of systems by decomposition and forward-backward elimination, drivers
- Least squares, QR decomposition
- Eigenvalue and eigenvector problems: all the EV or in a subset
- Generalized eigenproblems: $Ax = \lambda Bx$

The different steps of initialization (MPI, grid, arrays descriptors, distribution of them) or of destruction (grid release, terminate BLACS) are the same as for PBLAS

Functions calls similar to LAPACK

Example: driver for $Ax = b$, A general

`dgsev_(n, nrhs, a, lda, ipiv, b, ldb, info)`

`pdgsev_(n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info)`

⇒ does the local part of the decomposition or of the forward-backward elimination; requires data transfers between processes (example: pivot in ipiv local to another process).

Must be called by all the processes of the grid

7.2 Sparse matrices: iterative methods

Same methods as for the shared memory machines

⇒ essential operations $y \leftarrow A * x$, $y \leftarrow \alpha x + y$

Data distributed among different processors: communication problems

⇒ do not spend more time in communication than in computation!

Example: $y = A * x$

Consider the case of a dense matrix of order $n = 4 * m$

distribution onto $p = 4$ processors?

$\mathbf{a}^{(j)}, 1 \leq j \leq n$: column vectors

$$A = (\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \dots, \mathbf{a}^{(n)})$$

$$\mathbf{y} = \sum_{i=1}^n x_i \mathbf{a}^{(i)} = \mathbf{y}^{(1)} + \mathbf{y}^{(2)} + \mathbf{y}^{(3)} + \mathbf{y}^{(4)}$$

$$\mathbf{y}^{(i)} = \sum_{i=1}^{n/4} x_i \mathbf{a}^{(i)}$$

⇒ Partition A into 4 blocks of m columns and \mathbf{x} into 4 blocks of m components

$$A = (A^{(1)}, A^{(2)}, A^{(3)}, A^{(4)}), \quad \mathbf{x} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)})$$

One distribute $A^{(j)}$ et $\mathbf{x}^{(j)}$ on the processor nb j and one computes in parallel the vectors $\mathbf{y}^{(j)}$, then, by reduction, one does their sum

Remark: One stores $(A^{(j)})^t$ for accessing the column element contiguously in C

For a sparse matrix, one proceeds in the same way but with a Morse structure "column oriented"

ab : non zero values stored by columns

il : row index,

$i_ad[j]$: adress of the beginning of the j -th column

$ab[nn]$	a_{11}	a_{21}	a_{22}	a_{32}	a_{33}	a_{24}	..
$il[nn]$	1	2	2	3	3	2	..
$i_ad[n]$	0	2	4	5	..		

Example: $C = A * B$

$$C = (\mathbf{c}^{(1)}, \mathbf{c}^{(2)}, \dots, \mathbf{c}^{(n)}), \quad B = (\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \dots, \mathbf{b}^{(n)})$$

$$\mathbf{c}^{(j)} = A \mathbf{b}^{(j)}, 1 \leq j \leq n$$

⇒ B is partitioned by rows

$$B = \begin{pmatrix} B[1] \\ B[2] \\ B[3] \\ B[4] \end{pmatrix}$$

and $A^{(j)}$ et $B^{[j]}$ are distributed on the processor nb j

Existing software: use PETSC

7.3 Batch systems

Software for managing batch jobs allowing to visualize the job progression

Jobs chaining

Principle: submit a sequence of shell commands (script)

Main softwares: LSF, (Sun) Grid Engine, PBS

Example: Grid Engine

Commands:

- *qsub script_file*

- *qstat [-f]*

- *qmon*

Script example

```
#!/bin/csh -f

# # (c) 2000 Sun Microsystems, Inc. #

# _____

# The shell used to run the script (shall be the same as the user login shell)

#$ -S /bin/bash

#Everything is looked for in the current directory if no special path is given

#$ -cwd

#Job name

#$ -N xhpl

#Error file

#$ -e xhpl.out

#Output file
```

```
#$ -o xhpl.out

# pe request (mpich queue)

#$ -pe mpich 18

# SLbench2

/opt/mpich/1.2.5.2/gcc/x86_64/bin/mpirun -np 18 -machinefile \

$HOME/.machines xhpl
```

Exemple: PBS

Main commands identical to SGE

Script example

```
#!/bin/sh

#

#Job name

#PBS -N "xhpl"

#

# Queue to submit to

#PBS -q dque@malc2

#
```

```
# Job Limits

#PBS -l cput=300

#

# Keep stout,-error in

#PBS -k eo

#

# Merge std error to stdout

#PBS -j oe

#

# SLbench2

/usr/bin/mpirun.mpich -machinefile $HOME/mpich/machines.single -np 4

$HOME/SLBENCH2/hpl/bin/xhpl > $HOME/SLBENCH2/hpl/bin/HPL.out

exit
```