



**Universidad Autónoma
De Nuevo León
Facultad de Ingeniería Mecánica
Y Eléctrica**



Materia

Lab. Sistemas Distribuidos y Paralelos

Tema

Sistemas paralelos

Profesor

Ing. Raquel Martínez

Alumnos

Matriculas

Victor Manuel Martínez Moreno	1522026
Kevin Michelle Contreras González	1500340
Diana Anaid Loza Cerda	1521190

Grupo: 306

Salón: 4-200

Hora: V5-V6

ScaLAPACK:

ScaLAPACK (Scalable LAPACK) es una librería que incluye un subconjunto de rutinas LAPACK rediseñado para computadores paralelos de memoria distribuida MIMD. Actualmente está escrito en estilo Single-Program-Multiple-Data usando pasos de mensaje explícitos para la comunicación entre procesos. Se asume que las matrices se colocan siguiendo una descomposición cíclica de bloque de dos dimensiones. ScaLAPACK está diseñada para computación heterogénea y es portable a cualquier computador que soporte MPI o PVM.

Al igual que LAPACK, las rutinas de ScaLAPACK se basan en algoritmos de particiones en bloque con el fin de minimizar la frecuencia de movimiento de datos entre los distintos niveles de la jerarquía de memoria. (Para este tipo de máquinas, la jerarquía de memoria incluye la memoria off-processor de otros procesadores, además de la jerarquía de los registros, la caché y la memoria local en cada procesador). Los bloques de construcción fundamentales de la biblioteca ScaLAPACK son versiones de memoria distribuida (PBLAS) de nivel 1, 2 y 3 de BLAS, y un conjunto de Basic Linear Algebra Comunicación Subprograms (BLACS) para las tareas de comunicación que surgen con frecuencia en cálculos paralelos de álgebra lineal. En las rutinas de ScaLAPACK, todas las comunicaciones entre procesos se producen dentro de PBLAS y BLACS. Uno de los objetivos de diseño de ScaLAPACK fue que las rutinas de ScaLAPACK se parezcan a sus equivalentes de LAPACK tanto como sea posible.

Instalación:

Para instalarlo en Linux se debe bajar el paquete scalapack_installer.tgz y para Windows se da una guía para usar las librerías en Visual Studio.

Paso 1: Descargar el scalapack-2.0.1.tgz y descomprimirlo

```
# cd /usr/local # wget http://www.netlib.org/scalapack/scalapack-2.0.1.tgz
# tar xvf scalapack-2.0.1.tgz
```

Paso 2: Se crea un enlace entre ficheros

```
# ln -s scalapack-2.0.1 scalapack
```

Paso 3: Entramos en la carpeta para configurar el SLmake.inc

```
# cd scalapack
# cp SLmake.inc.example SLmake.inc
# vim SLmake.inc
    BLASLIB      = -lblas -L/usr/local/BLAS
    LAPACKLIB    = -llapack -L/usr/local/LAPACK
    LIBS         = $(LAPACKLIB) $(BLASLIB)
# make
# export SCALAPACK=/usr/local/scalapack/libscalapack.a
```

```

* =====
*
* .. Parameters ..
INTEGER      BLOCK_CYCLIC_2D, CSRC_, CTXT_, DLEN_, DTYPE_,
$            LLD_, MB_, M_, NB_, N_, RSRC_
PARAMETER    ( BLOCK_CYCLIC_2D = 1, DLEN_ = 9, DTYPE_ = 1,
$            CTXT_ = 2, M_ = 3, N_ = 4, MB_ = 5, NB_ = 6,
$            RSRC_ = 7, CSRC_ = 8, LLD_ = 9 )
INTEGER      NIN
PARAMETER    ( NIN = 11 )
DOUBLE PRECISION ONE, ZERO
PARAMETER    ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*
* .. Local Scalars ..
LOGICAL      ISIOPROCESSOR
INTEGER      CSRC, I, ICTXT, IEND, ISIZE, ISTART, J, JEND,
$            JSIZE, JSTART, LDD, LWORK, M, MB, MM, MYCOL,
$            MYROW, N, NB, NN, NPCOL, NPROW, RSRC
DOUBLE PRECISION ALPHA, BETA
*
* .. Local Arrays ..
INTEGER      DESCWORK( DLEN_ ), IWORK( 2 )
*
* .. External Subroutines ..
EXTERNAL     BLACS_GRIDINFO, DESCSET, IGEBR2D, IGEBS2D,
$            PDGEADD, PXERBLA
*
* .. Intrinsic Functions ..
INTRINSIC    INT, MAX, MIN
*
* .. Executable Statements ..
LWORK = DESCA( MB_ )
ICTXT = DESCA( CTXT_ )
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
ISIOPROCESSOR = ( ( MYROW.EQ.IRREAD ) .AND. ( MYCOL.EQ.ICREAD ) )
*
IF( ISIOPROCESSOR ) THEN
  OPEN( NIN, FILE = FILENAM, STATUS = 'OLD', FORM = 'FORMATTED',
$      ACCESS = 'SEQUENTIAL', ERR = 50 )
  -----

```

```

      MB = MM
      NB = NN
      RSRC = IRREAD
      CSRC = ICREAD
      LDD = MAX( 1, MM )
      CALL DESCSET( DESCWORK, MM, NN, MB, NB, RSRC, CSRC, ICTXT, LDD )
*
      DO 40 JSTART = 1, N, NN
        JEND = MIN( N, JSTART+NN-1 )
        JSIZE = JEND - JSTART + 1
        DO 30 ISTART = 1, M, MM
          IEND = MIN( M, ISTART+MM-1 )
          ISIZE = IEND - ISTART + 1
          ALPHA = ONE
          BETA = ZERO
          IF( ISIOPROCESSOR ) THEN
            DO 20 J = 1, JSIZE
              DO 10 I = 1, ISIZE
                READ( NIN, FMT = *, ERR = 50 )WORK( I+( J-1 ) *
$                  LDD )
10              CONTINUE
20              CONTINUE
                END IF
*
                CALL PDGEADD( 'NoTrans', ISIZE, JSIZE, ALPHA, WORK, 1, 1,
$                  DESCWORK, BETA, A, ISTART, JSTART, DESCA )
30            CONTINUE
40          CONTINUE
          IF( ISIOPROCESSOR ) THEN
            CLOSE ( NIN, ERR = 50 )
          END IF
          WORK( 1 ) = DESCA( MB_ )
          RETURN
50        CONTINUE
        CALL PXERBLA( DESCA( CTXT_ ), 'PLAWRITE', 1 )
        WORK( 1 ) = DESCA( MB_ )
        RETURN
*
*      End of PDLAREAD
*
      END

```

BLAS

Los subprogramas de álgebra lineal básica (BLAS) definen un conjunto de operaciones fundamentales sobre vectores y matrices el cual puede ser utilizado para crear la funcionalidad de álgebra lineal de alto nivel optimizado.

La biblioteca proporciona una capa de bajo nivel del que se corresponde directamente con el estándar BLAS lenguaje C, denominado aquí como "CBLAS", y una interfaz de alto nivel para las operaciones sobre vectores y matrices de GSL. Los usuarios que estén interesados en las operaciones simples sobre GSL vectoriales y matriciales objetos deben usar la capa de alto nivel se describe en este capítulo. Las funciones se declaran en el archivo `gsl_blas.h` y deben satisfacer las necesidades de la mayoría de los usuarios.

Tenga en cuenta que las matrices de GSL se implementan una densa-almacenamiento utilizando lo que la interfaz sólo incluye las funciones BLAS densa de almacenamiento correspondiente. La funcionalidad completa BLAS para la banda-formato y matrices en formato empaquetado está disponible a través de los CBLAS interfaz de bajo nivel. Del mismo modo, los vectores de GSL se limitan a pasos positivos, mientras que la interfaz de bajo nivel apoya CBLAS zancadas negativos como se especifica en la standard.12

BLAS

La interfaz para la capa de `gsl_cblas` se especifica en el archivo de `gsl_cblas.h`. Esta interfaz se corresponde con el estándar del Foro Técnico BLAS para la interfaz de C al legado BLAS Implementaciones. Los usuarios que tienen acceso a otros conforme CBLAS implementaciones pueden usar estos en lugar de la versión proporcionada por la biblioteca. Tenga en cuenta que los usuarios que tienen sólo unas bibliotecas Fortran BLAS pueden utilizar un envoltorio conformant CBLAS para convertirlo en una biblioteca CBLAS. Existe una CBLAS envoltorio de referencia para heredados Fortran Implementaciones como parte de la norma CBLAS y se pueden obtener a partir de Netlib. El conjunto completo de funciones CBLAS aparece en un apéndice (véase GSL CBLAS Library).

Instalación

Para poder instalar los componentes necesario de esta librería desde Linux o cualquier otra distribución se necesitan instalar los complementos completos de LAPACK y ATLAS

Paso 1: extracción

```
$ tar xzf blas.tgz
```

Esto creará un directorio BLAS.

Paso 2: Compilación

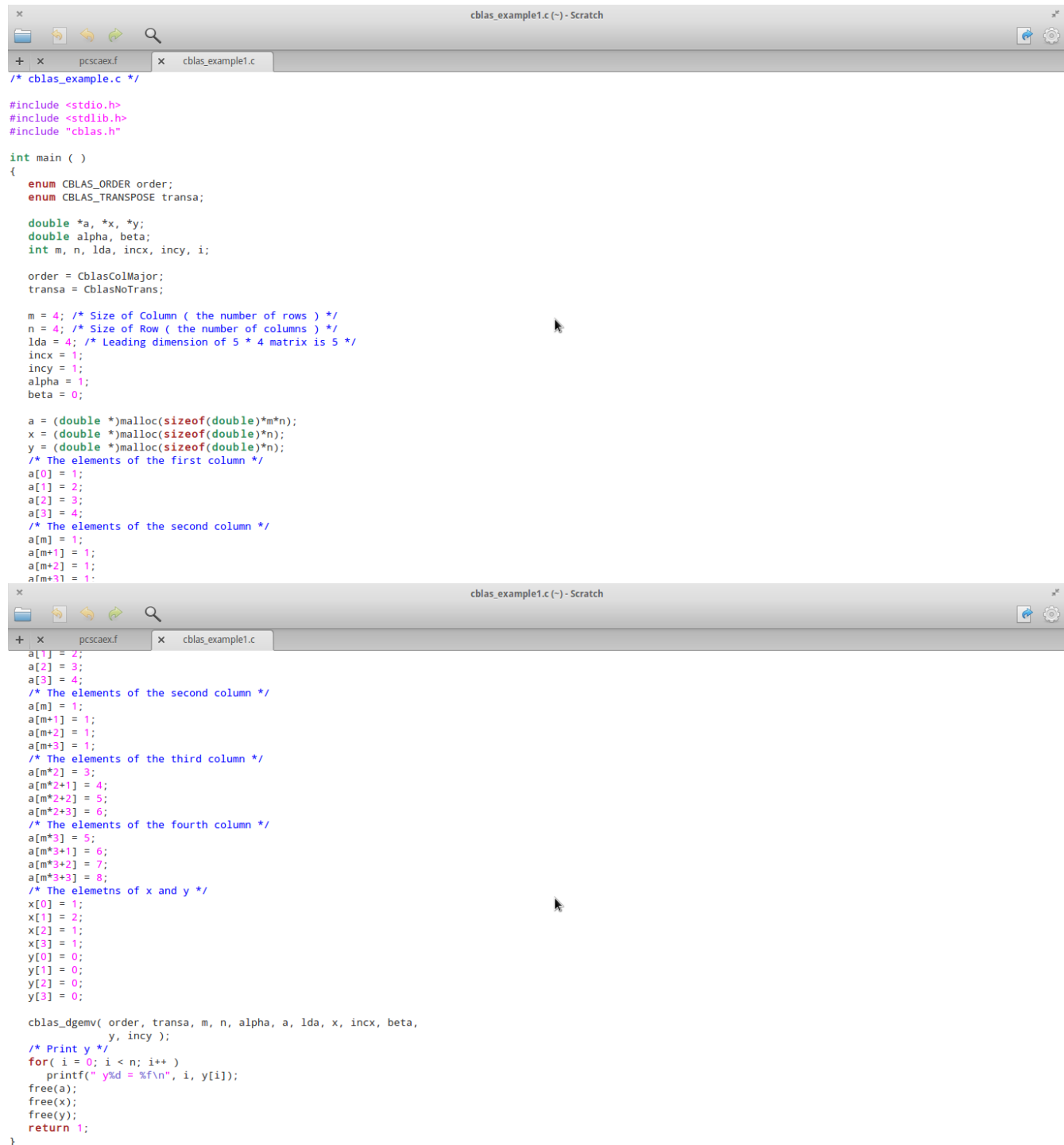
```
$ cd BLAS
```

```
$ make all
```

El paquete es un montón de archivos de Fortran, el cual recoge a una biblioteca : si tiene que personalizar algo, el lugar correcto es el `make.inc` archivo : normalmente se podía personalizar el nombre de la biblioteca (la variable se llama `PLAT` , su valor por defecto es `blas_LINUX.a`) y el compilador Fortran (por ejemplo `FORTTRAN = gfortran`)

Paso 3 : Configuración

Si todo fue correcto en el paso anterior, la biblioteca BLAS está en el directorio , llamado blas_LINUX.a : puede copiarlo donde usted prefiere, usted apenas tiene que establecer la ruta correcta cuando desee utilizarlo.



```
/* cblas_example.c */

#include <stdio.h>
#include <stdlib.h>
#include "cblas.h"

int main ( )
{
    enum CBLAS_ORDER order;
    enum CBLAS_TRANSPOSE transa;

    double *a, *x, *y;
    double alpha, beta;
    int m, n, lda, incx, incy, i;


    order = CblasColMajor;
    transa = CblasNoTrans;

    m = 4; /* Size of Column ( the number of rows ) */
    n = 4; /* Size of Row ( the number of columns ) */
    lda = 4; /* Leading dimension of 5 * 4 matrix is 5 */
    incx = 1;
    incy = 1;
    alpha = 1;
    beta = 0;

    a = (double *)malloc(sizeof(double)*m*n);
    x = (double *)malloc(sizeof(double)*n);
    y = (double *)malloc(sizeof(double)*n);
    /* The elements of the first column */
    a[0] = 1;
    a[1] = 2;
    a[2] = 3;
    a[3] = 4;
    /* The elements of the second column */
    a[m] = 1;
    a[m+1] = 1;
    a[m+2] = 1;
    a[m+3] = 1;
    /* The elements of the third column */
    a[m*2] = 3;
    a[m*2+1] = 4;
    a[m*2+2] = 5;
    a[m*2+3] = 6;
    /* The elements of the fourth column */
    a[m*3] = 5;
    a[m*3+1] = 6;
    a[m*3+2] = 7;
    a[m*3+3] = 8;
    /* The elements of x and y */
    x[0] = 1;
    x[1] = 2;
    x[2] = 1;
    x[3] = 1;
    y[0] = 0;
    y[1] = 0;
    y[2] = 0;
    y[3] = 0;

    cblas_dgemv( order, transa, m, n, alpha, a, lda, x, incx, beta,
                y, incy );
    /* Print y */
    for( i = 0; i < n; i++ )
        printf(" y%d = %f\n", i, y[i]);
    free(a);
    free(x);
    free(y);
    return 1;
}
```

```
manuel@moreno: ~  
+ x manuel@moreno: ~  
manuel@moreno:~$ cc cblas_example1.c -llapack -lblas -lm  
manuel@moreno:~$ ./a.out  
y0 = 11.000000  
y1 = 14.000000  
y2 = 17.000000  
y3 = 20.000000  
manuel@moreno:~$
```



LAPACK

LAPACK está escrito en Fortran 90 y proporciona rutinas para resolver sistemas de ecuaciones lineales simultáneas, por mínimos cuadrados de soluciones de sistemas de ecuaciones lineales, problemas de valores propios y problemas de valores singulares. Las factorizaciones matriciales asociadas (LU, Cholesky , QR, SVD , Schur , Schur generalizado) también se proporcionan, así como los cálculos relacionados, como la reordenación de las factorizaciones de Schur y la estimación de números de condición. Matrices densas y de bandas se manejan, pero no generales matrices dispersas. En todas las áreas, se proporciona una funcionalidad similar para matrices reales y complejos, tanto en simple y doble precisión.

El objetivo original del proyecto LAPACK era hacer el EISPACK ampliamente utilizado y bibliotecas Linpack ejecutar de manera eficiente el vector de memoria compartida y procesadores en paralelo. LAPACK aborda este problema mediante la reorganización de los algoritmos de utilizar operaciones de la matriz de bloques, como la multiplicación de matrices, en los lazos más íntimos. Estas operaciones de bloque pueden ser optimizados para cada arquitectura para dar cuenta de la jerarquía de memoria, y así proporcionan una forma transportable para lograr una alta eficiencia en diversas máquinas modernas. LAPACK requiere que las operaciones de la matriz de bloques altamente optimizado ser ya aplicadas en cada máquina.

Las rutinas LAPACK se han escrito para que la mayor cantidad posible del cálculo se realiza mediante llamadas a los subprogramas de álgebra lineal básica (BLAS) . LAPACK está diseñado desde el principio para aprovechar las del nivel 3 de BLAS - un conjunto de especificaciones para los subprogramas Fortran que hacen varios tipos de multiplicación de la matriz y la solución de sistemas triangulares con múltiples lados derechos. Debido a la granularidad gruesa de las operaciones de nivel 3 de BLAS, su uso promueve una alta eficiencia en muchas computadoras de alto rendimiento, sobre todo si las implementaciones especialmente codificados son proporcionados por el fabricante.

Instalación.

Paso 1: instalar las siguientes dependencias:

```
$ sudo apt-get install build-essential lapack3-dev  
refblas3-dev atlas3-base-dev checkinstall
```

Una vez hecho esto nos descargamos las librerías Lapack++ en nuestro home: lapack-3.5.0.tgz.

Paso 2: descomprimos el paquete:

```
$ tar xzf lapack-3.5.0.tgz
```

Paso 3: Accedemos via Nautilus a la nueva carpeta lapack-3.5.0, abrimos la terminal y escribimos:

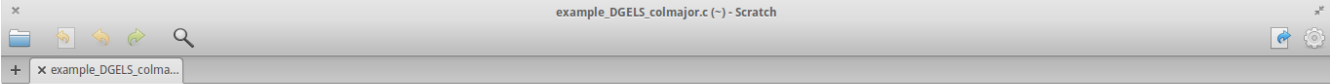
```
$ ./configure  
$ make
```



```
$ sudo checkinstall
$ sudo cp /usr/local/lib/liblapackpp.so.5 /usr/lib/
```

Ya lo esta instalado, para compilar el programa que hayas realizado hacer:

```
$ g++ miprograma.cpp -o miprograma -llapackpp
```



```
/* Includes */
#include <stdio.h>
#include <lapacke.h>
#include "lapacke_example_aux.h"

/* Main program */
int main (int argc, const char * argv[])
{
    /* Locals */
    double A[5][3] = {1,2,3,4,5,1,3,5,2,4,1,4,2,5,3};
    double b[5][2] = {-10,12,14,16,18,-3,14,12,16,16};
    lapack_int info,m,n,lda,ldb,nrhs;
    int i,j;

    /* Initialization */
    m = 5;
    n = 3;
    nrhs = 2;
    lda = 5;
    ldb = 5;

    /* Print Entry Matrix */
    print_matrix_colmajor( "Entry Matrix A", m, n, *A, lda );
    /* Print Right Rand Side */
    print_matrix_colmajor( "Right Hand Side b", n, nrhs, *b, ldb );
    printf( "\n" );

    /* Executable statements */
    printf( "LAPACKE_dgels (col-major, high-level) Example Program Results\n" );
    /* Solve least squares problem*/
    info = LAPACKE_dgels(LAPACK_COL_MAJOR,'N',m,n,nrhs,*A,lda,*b,ldb);

    /* Print Solution */
    print_matrix_colmajor( "Solution", n, nrhs, *b, ldb );
    printf( "\n" );
    exit( 0 );
} /* End of LAPACKE_dgels Example */
```