Lecture – HPC Libraries

HPC

Assoc.Prof. Dr. Sascha Hunold

TU Wien

2024-01-16

Outline

- LINPACK
- Libraries for Scientific Computing

LINPACK / HPLinpack / HPL

LINPACK Benchmark

- Jack J. Dongarra, Piotr Luszczek and Antoine Petitet, The LINPACK Benchmark: past, present and future [3]
- "The first 'LINPACK Benchmark' report appeared as an appendix in the LINPACK Users' Guide in 1979".
- originally designed to provide execution times required to solve linear equations (for LINPACK package users)
- benchmark reports the performance for solving a general dense matrix problem $Ax = b$ in 64-bit FP arithmetic

Benchmark	Matrix dimension	Optimizations allowed	Parallel processing
LINPACK 100	100	Compiler	Compiler parallelization
LINPACK 1000	1000	Manual	Multiprocessor implementations
LINPACK Parallel	1000	Manual	Yes
HPLinpack	Arbitrary	Manual	Yes

Original LINPACK Benchmark

- used matrix size 100 due to memory limitation of computers in 1979
- such matrices had 10,000 FP elements, were large enough
- algorithm
  - LU decomposition with partial pivoting (for numerical stability)
  - matrix type is real, general and dense
  - elements randomly generated in $[-1, 1]$
  - solving the system requires $O(n^3)$ FP operations
  - or approximately $\frac{2}{3}n^3 + O(n^2)$

LINPACK in detail

$Ax = b$, $A \in R^{n \times n}$, $x, b \in R^n$

$$L = \begin{pmatrix} l_{1,1} & 0 & \dots & 0 & 0 \\ l_{2,1} & l_{2,2} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ l_{n-1,1} & l_{n-1,2} & \dots & l_{n-1,n-1} & 0 \\ l_{n,1} & l_{n,2} & \dots & l_{n,n-1} & l_{n,n} \end{pmatrix}$$

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & \dots & u_{1,n-1} & u_{1,n} \\ 0 & u_{2,2} & \dots & u_{2,n-1} & u_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & \dots & 0 & u_{n,n} \end{pmatrix}$$

- $Ax = LUx = b$
- solving
  - $Ly = b$ by forward substitution
  - $Ux = y$ solved by back-substitution

LU Factorization - A Reminder

LU once again

```
A=rand(5,5);
[L,U,P]=lu(A)
```

```
L =

   1.0000        0        0        0        0
   0.9070   1.0000        0        0        0
   0.9944  -0.1090   1.0000        0        0
   0.1064  -0.5410   0.8667   1.0000        0
   0.9289   0.8595   0.6635  -0.4975   1.0000

U =

   0.7153   0.8796   0.1404   0.0818   0.9316
        0  -0.4628   0.2619   0.5573   0.0832
        0        0   0.2811   0.3869  -0.7673
        0        0        0   0.4650   0.8997
        0        0        0        0   0.9176

P =

Permutation Matrix

   0   0   1   0   0
   0   0   0   0   1
   1   0   0   0   0
   0   1   0   0   0
   0   0   0   1   0
```

LU once again

```
1 A = rand(3,3)
2 [L,U,P]=lu(A);
3 P*A
4 L*U
```

```
A =

   0.1115   0.7903   0.5738
   0.8870   0.4453   0.1680
   0.3795   0.6725   0.1982

ans =

   0.8870   0.4453   0.1680
   0.1115   0.7903   0.5738
   0.3795   0.6725   0.1982

ans =

   0.8870   0.4453   0.1680
   0.1115   0.7903   0.5738
   0.3795   0.6725   0.1982
```

LU .. and once again

```
octave:69> A = rand(4,4)
A =

   0.81041   0.68261   0.20580   0.15378
   0.50094   0.91942   0.30133   0.12460
   0.75433   0.44770   0.65324   0.22037
   0.94289   0.89075   0.64287   0.72682

octave:70> b = [ 1; 2; 3; 4; ]
b =

   1
   2
   3
   4

octave:71> A \ b
ans =

  -1.1348
   1.0574
   4.6495
   1.5672
```

```
octave:73> [L,U,P] = lu(A)
octave:74> y = L \ (P*b)
y =

   4.00000
  -0.12512
  -2.46124
  -1.07378

octave:75> x = U \ y
x =

  -1.1348
   1.0574
   4.6495
   1.5672
```

LU - by hand

- solve U and then compute L using backward substitution

$$\begin{pmatrix} 3 & 5 & 10 \\ 2 & 2 & 3 \\ 1 & 8 & 5 \end{pmatrix} \begin{matrix} \\ \leftarrow\!\lrcorner\, -\frac{2}{3} \\ \leftarrow\!\!\!\!\!\!\!\!\!\!\lrcorner\, -\frac{1}{3} \end{matrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{1}{3} & TBC & 1 \end{pmatrix} \cdot \begin{pmatrix} 3 & 5 & 10 \\ 0 & -\frac{4}{3} & \frac{-11}{3} \\ 0 & \frac{19}{3} & \frac{5}{3} \end{pmatrix} \begin{matrix} \\ \leftarrow\!\lrcorner\, \frac{19}{4} \\ \\ \end{matrix}$$

$$\begin{pmatrix} 3 & 5 & 10 \\ 2 & 2 & 3 \\ 1 & 8 & 5 \end{pmatrix} \begin{matrix} \\ \leftarrow\!\lrcorner\, -\frac{2}{3} \\ \leftarrow\!\!\!\!\!\!\!\!\!\!\lrcorner\, -\frac{1}{3} \end{matrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{1}{3} & -\frac{19}{4} & 1 \end{pmatrix} \cdot \begin{pmatrix} 3 & 5 & 10 \\ 0 & -\frac{4}{3} & \frac{-11}{3} \\ 0 & 0 & -\frac{63}{4} \end{pmatrix}$$

LU - as always – verify!

```
L = [ 1 0 0; 2/3 1 0 ; 1/3 -19/4 1 ]
U = [ 3 5 10; 0 -4/3 -11/3; 0 0 -63/4 ]
L*U
```

```
L =

   1.0000         0         0
   0.6667    1.0000         0
   0.3333   -4.7500    1.0000

U =

    3.0000    5.0000   10.0000
         0   -1.3333   -3.6667
         0         0  -15.7500

ans =

    3    5   10
    2    2    3
    1    8    5
```

HPLinpack

- rules:
  - solve systems of equations allowing to vary problem size n
  - measure execution time for each problem size
  - base FP execution rate on $2n^3/3 + 2n^2$ operations independent of actual method
  - compute and report residuals
- compute residuals
  - ϵ relative machine precision, problem size n
  - $r = \frac{||Ax-b||_\infty}{\epsilon \cdot n \cdot ||x||_\infty \cdot ||A||_\infty \cdot ||b||_\infty}$
    - numerically correct if in $O(1)$
  - $||x||_\infty = \max_{i=1,\ldots,n} |x_i|$
  - $||A||_\infty = \max_{i=1,\ldots,m} \sum_{j=1}^{n} |a_{ij}|$
  - "The relative machine precision usually the smallest positive number such that fl(1.0 - eps) < 1.0, where fl denotes the computed value and eps is the relative machine precision."
    `http://www.netlib.org/benchmark/linpackjava/`

Residuals for our example I

```
A = [ 3 5 10; 2 2 3; 1 8 5 ]
b = [ 1 ; 2 ; 3]
disp(eps)

n = 3
format long
x = A\b
```

```
A =

    3    5   10
    2    2    3
    1    8    5
b =

   1
   2
   3
2.220446049250313e-16
octave> n = 3
octave> x =

   1.285714285714286
   0.571428571428571
  -0.571428571428572
```

Residuals for our example II

```
norm(A*x-b,"inf") / (eps*n*norm(x,"inf")*norm(A,"inf")*norm(b,"inf"))

# manual modifactions of result will decrease precision
x = [ 1.285714285 ; 0.571428571 ; -0.571428571 ]
norm(A*x-b,"inf") / (eps*n*norm(x,"inf")*norm(A,"inf")*norm(b,"inf"))
```

```
ans = 1.920438957475994e-02
octave> octave> x =

   1.285714285000000
   0.571428571000000
  -0.571428571000000
ans = 43244.44446846913
```

HPL

HPL

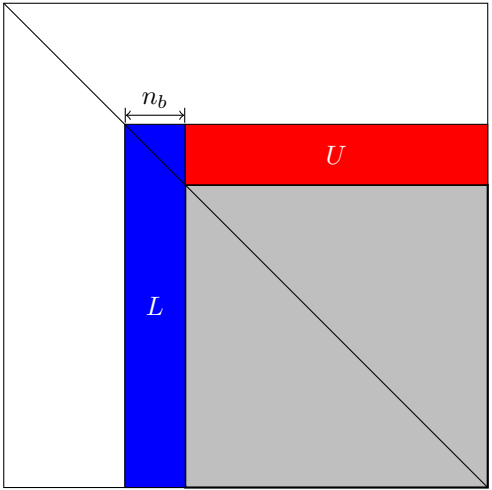
- reference implementation of HPLinpack (High Performance LINPACK)
  - see `http://www.netlib.org/benchmark/hpl/` and `http://icl.eecs.utk.edu/hpl/`
- $r \times c$ processor grid
- matrices distributed using 2D block-cyclic data distribution
  - example, 2×2 processors, number of subblocks is 4 ($n/n_b = 4$)
  - n_b is the blocking factor

$$\left(\begin{array}{c|c|c|c} P_0 & P_1 & P_0 & P_1 \\ \hline P_2 & P_3 & P_2 & P_3 \\ \hline P_0 & P_1 & P_0 & P_1 \\ \hline P_2 & P_3 & P_2 & P_3 \end{array}\right)$$

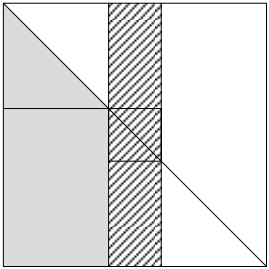
- have a look here: `http://acts.nersc.gov/scalapack/hands-on/datadist.html`

```
1 MPI_Barrier(...); /* All the nodes start at the same time */
2 HPL_ptimer(...);  /* Start wall-clock timer. */
3 HPL_pdgesv(...);  /* Solve system of equations. */
4 HPL_ptimer(...);  /* Stop wall-clock timer. */
5 MPI_Reduce(...);  /* Obtain the maximum wall-clock time. */
```

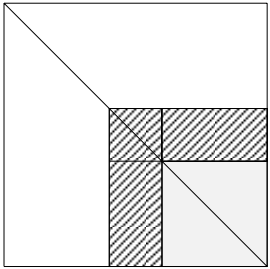
- solve $Ax = b$
  1. compute LU factorization with row partial pivoting of $n \times (n + 1)$ coefficient matrix, $[Ab] = [LUy]$
  2. x is obtained by solving upper triangular system $Ux = y$, L is applied to b during factorization
- L stays unpivoted and array of pivots is not returned
- main loop of factorization: right-looking variant
  - in each iteration n_b columns are factorized (block-oriented)
  - trailing submatrix is updated
  - computation logically partitioned with same block size n_b that was used for 2D data distribution


n_b
U
L

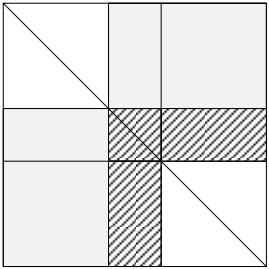
- per iteration we need to factor panel of column of A
- this panel is owned by a column of processes
- the panel factorization is done recursively
  - recursion stopping at: user-defined number of columns (NBMIN)
- 3 different choices for factorizing panel
  1. Crout
  2. left-looking
  3. right-looking
- for each panel column
  - pivot
  - associated swap
  - and broadcast of pivot row
  - all in one single communication step
  - binary-exchange (leave-on-all) reduction performs the three operations at once



Left-looking LU



Right-looking LU

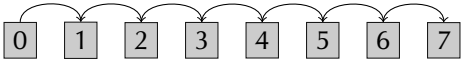


Crout LU

source: `http://www.netlib.org/ddsv/figures/fig5-2.ps`

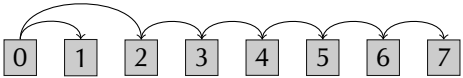
- after panel factorization is done, we need to broadcast panel to other process columns
- several broadcast algorithms available
  1. Increasing-ring
  2. Increasing-ring (modified)
  3. Increasing-2-ring
  4. Increasing-2-ring (modified)
  5. Long (bandwidth reducing)
  6. Long (bandwidth reducing modified)
- modified variants will make the next processor (that participates in factorization of next panel) not send messages (only receive)

- Increasing-ring



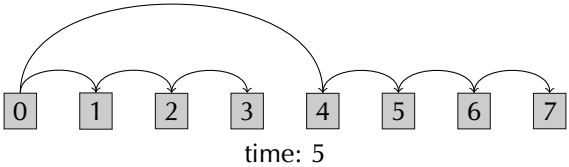
time: 7

- Increasing-ring (modified)
  - processor 1 only needs to receive data (which is the next one to compute)
  - often broadcast of choice

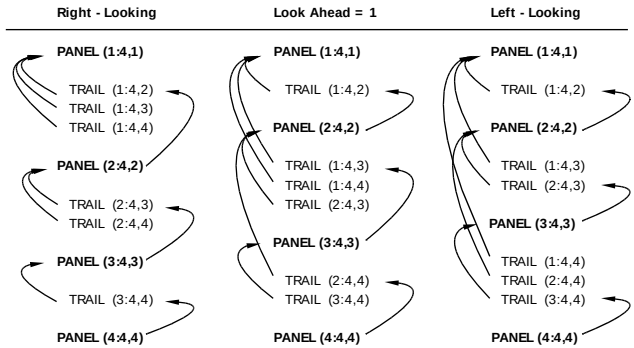


time: 7

- Increasing-2-ring
  - divide Q processes into two parts: $0 \rightarrow 1$ and $0 \rightarrow Q/2$
  - 1 and $Q/2$ then source of two rings



HPL – Look-ahead illustrated



- "variants of block LU factorization with 1D block cyclic work partitioning for a problem with the coefficient matrix of size 4 by 4 blocks"
- source: J. Kurzak and J. Dongarra, Implementing linear algebra routines on multi-core processors with pipelining and a look ahead. LAPACK Working Note 178, September 2006

HPL.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
4            # of problems sizes (N)
29 30 34 35  Ns
4            # of NBs
1 2 3 4      NBs
0            PMAP process mapping (0=Row-,1=Column-major)
3            # of process grids (P x Q)
2 1 4        Ps
2 4 1        Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

Libraries for Scientific Computing

BLAS

BLAS

- Basic Linear Algebra Subprograms (BLAS)
- C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage. *ACM Trans. Math. Soft*, 5:308–323, 1979. [5]
- "BLAS interface supports portable high-performance implementation of applications that are matrix and vector computation intensive" [4]
- architecture-specific optimization left to an expert
- first BLAS interface from the 1970s
  - vector computer widely used
- now referred to as Level-1 BLAS
  - `?axpy` operation $y = \alpha x + y$
  - `?dot` (inner product, dot product) $s = x \cdot y = \sum_{i=0}^{n-1} x_i y_i$

```
1 #pragma omp parallel for
2 for(i=0; i<N; i++) {
3   y[i] += alpha * x[i]
4 }
```

Level-1 BLAS

- perform $O(n)$ operations on $O(n)$ data
- `_axpy( n, alpha, x, incx, y, incy )`
- "_" indicates data type

`s`	single precision
`d`	double precision
`c`	single precision complex
`z`	double precision complex

- axpy: alpha *times* x *plus* y
- parameters
  - n number of elements
  - alpha scalar
  - x and y, memory locations
  - increment to locate the elements of vectors

Level-1 BLAS

- examples of Level-1 BLAS routines:

Routine	Operation
`_swap`	$x \leftrightarrow y$
`_scal`	$x \leftarrow \alpha x$
`_copy`	$y \leftarrow x$
`_axpy`	$y \leftarrow \alpha x + y$
`_dot`	$x^T y$
`_nrm2`	$\|\|x\|\|_2$

Level-2 BLAS

- involve $O(n^2)$ operations on $O(n^2)$ data
- matrix-vector operations, e.g., matrix-vector product: $y \leftarrow Ax$
  - x, y are vectors, A is a matrix
- naming convention for Level-2 BLAS `_XXYY`
- "`_`" is data type
- XX indicates shape of matrix

XX	matrix shape
`ge`	general (rectangular)
`sy`	symmetric
`he`	Hermitian
`tr`	triangular

- YY specifies operation

YY	operation
`mv`	matrix vector multiplication
`sv`	solve vector
`r`	rank-1 update
`r2`	rank-2 update

Level-2 BLAS

- some Level-2 BLAS operations

Routine	Operation	Details
`_gemv`	general matrix-vector multiplication	$y \leftarrow \alpha Ax + \beta y$
`_symv`	symmetric matrix-vector multiplication	
`_trmv`	triangular matrix-vector multiplications	$x \leftarrow Ax$, $x \leftarrow A^T x$, $x \leftarrow A^H x$
`_ger`	general rank-1 update	$A \leftarrow \alpha x y^T + A$
`_syr`	symmetric rank-1 update	$A \leftarrow \alpha x x^T + A$
`_syr2`	symmetric rank-2 update	$A \leftarrow \alpha x y^T + \alpha y x^T + A$

Let's play with _ger

- general rank-1 update _ger
  - $A \leftarrow \alpha x y^T + A$

```
A = [1,2,3;4,5,6;7,8,9]
x = [1;0;0]; # let's select first row
y = [0,2,0]; # add 2 in second column
x*y+A
```

```
A =

   1   2   3
   4   5   6
   7   8   9

ans =

   1   4   3
   4   5   6
   7   8   9
```

Level-3 BLAS

- involve $O(n^3)$ operations on $O(n^2)$ data

Routine	Operation	Details
`_gemm`	general matrix-matrix multiplication	$C \leftarrow \alpha AB + \beta C$
`_symm`	symmetric matrix-matrix multiplication	
`_trsm`	triangular solve with multiple right-hand sides	$B \leftarrow \alpha A^{-1} B$
`_syrk`	symmetric rank-k update	$C \leftarrow \alpha AA^T + \beta C$
`_syrk2`	symmetric rank-2k update	$C \leftarrow \alpha AB^T + \alpha BA^T + C$

Level-3 BLAS _syrk

- $C \leftarrow \alpha A A^T + \beta C$

```
C = [1,2,3;4,5,6;7,8,9]
A = [0,0,0;0,0,1;0,0,0]
alpha = 0.5;
C = alpha * A * A' + C
```

```
C =

   1   2   3
   4   5   6
   7   8   9

A =

   0   0   0
   0   0   1
   0   0   0

C =

   1.0000   2.0000   3.0000
   4.0000   5.5000   6.0000
   7.0000   8.0000   9.0000
```

- CBLAS
  - C interface for BLAS with support for row and column major matrices
- vendor implementations
  - IBM ESSL (Engineering and Scientific Subroutine Library)
  - Intel MKL (Intel® Math Kernel Library)
  - AMD ACML (Core Math Library)
  - NEC MathKeisan
  - HP MLIB (Math Library)
  - cuBLAS `https://developer.nvidia.com/cublas`
- open source implementations
  - ATLAS
  - GotoBLAS
  - OpenBLAS

BLAS Example – $A := \alpha x y^T + A$

```
1  program ger_main
2  real a(5,3), x(10), y(10), alpha
3  integer m, n, incx, incy, i, j, lda
4  m = 2
5  n = 3
6  lda = 5
7  incx = 2
8  incy = 1
9  alpha = 0.5
10 do i = 1, 10
11   x(i) = 1.0
12   y(i) = 1.0
13 end do
14 do i = 1, m
15   do j = 1, n
16     a(i,j) = j
17   end do
18 end do
19 ! perform the rank 1 operation
20 call sger (m, n, alpha, x, incx, y, incy, a, lda)
21 print*, 'Matrix A: '
22 do i = 1, m
23   print*, (a(i,j), j = 1, n)
24 end do
25 end
```

BLAS Example Output

- output:

```
Matrix A:

1.50000 2.50000 3.50000

1.50000 2.50000 3.50000
```

BLAS Example Output

- output:

```
Matrix A:

1.50000 2.50000 3.50000

1.50000 2.50000 3.50000
```

- let us verify with R

```r
A <- matrix(c(1,2,3, 1,2,3), nrow = 2, ncol = 3, byrow = TRUE)
x <- rep(1.0,10)
y <- rep(1.0,10)
alpha <- 0.5
outer(alpha*x,y)[c(1:2),c(1:3)] + A
```

```
     [,1] [,2] [,3]
[1,]  1.5  2.5  3.5
[2,]  1.5  2.5  3.5
```

LAPACK

LAPACK

- LAPACK (Linear Algebra PACKage) [1]
- library of Fortran 77 subroutines
- dense matrix computations
  - solve systems of linear equations
    - $Ax = b$
  - linear least square problems
    - minimize $||b - Ax||_2$
  - eigenvalue problems, $Ax = \lambda x$
  - singular value problems
  - matrix factorizations
    - e.g., LU, QR, Cholesky
- designed for high efficiency on vector processors, super-scalar workstations, shared-memory multiprocessors

LAPACK

- three types of routines
  1. driver routines
     - problem solving
     - sequence of computational routines
  2. computational routines
     - distinct computational tasks
  3. auxiliary routines
     - low-level computation

- source: http://www.netlib.org/lapack/lapacke.html

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

int main (int argc, const char * argv[])
{
   double a[5][3] = {1,1,1,2,3,4,3,5,2,4,2,5,5,4,3};
   double b[5][2] = {-10,-3,12,14,14,12,16,16,18,16};
   lapack_int info,m,n,lda,ldb,nrhs;
   int i,j;
   m = 5; n = 3; nrhs = 2; lda = 3; ldb = 2;
   info = LAPACKE_dgels(LAPACK_ROW_MAJOR,'N',m,n,nrhs,*a,lda,*b,ldb);

   for(i=0;i<n;i++)
   {
      for(j=0;j<nrhs;j++)
      {
         printf("%lf ",b[i][j]);
      }
      printf("\n");
   }
   return(info);
}
```

LAPACK

- solving the least squares solution to an over-determined system of linear equations
- multiple right-hand sides

```
gcc -o lapack1 lapack1.c -llapacke
./lapack1
2.000000 1.000000
1.000000 1.000000
1.000000 2.000000
```

- verifying using Octave

```
1 a = [1 1 1 ; 2 3 4 ; 3 5 2; 4 2 5 ; 5 4 3  ];
2 b = [-10 -3; 12 14; 14 12; 16 16; 18 16];
3 ols(b,a)
```

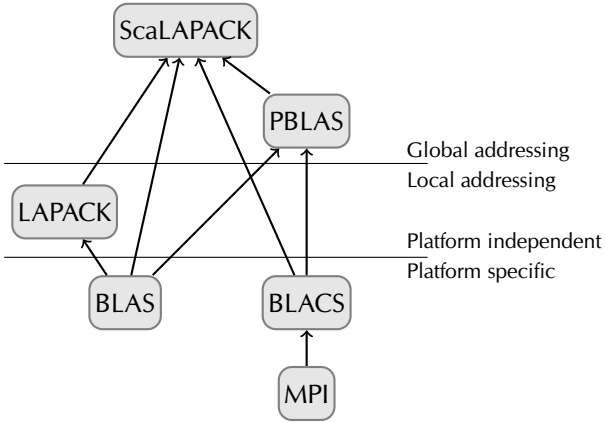
```
ans =

   2   1
   1   1
   1   2
```

ScaLAPACK

- Scalable Linear Algebra PACKage or Scalable LAPACK
- library for high-performance linear algebra [2]
- continuation of LAPACK for distributed-memory machines
  - not everything from LAPACK covered in ScaLAPACK
- based on message-passing (MPI)
- mostly written in Fortran 77 (with few exceptions)
- Single Program Multiple Data (SPDM) style with *explicit message passing*
- solves linear equations, linear least squares problems, eigenvalue problems, singular value problems
- block-partitioned algorithms to minimize the frequency of data movement between levels of the memory hierarchy

ScaLAPACK

- PBLAS, Parallel BLAS (Level 1-3)
  - e.g., matrix vector (p?gemv), matrix matrix (p?gemm), vector scalar (p?axpy)
- BLACS
  - communication tasks
  - implementations: MPI, IBM MPL, Intel NX, PVM
- dense and band matrices
  - but not general sparse matrices
- two-dimensional block cyclic distribution scheme


ScaLAPACK
PBLAS
Global addressing
Local addressing
LAPACK
Platform independent
Platform specific
BLAS
BLACS
MPI

References I

[1] J. Dongarra and P. Luszczek. “LAPACK”. In: *Encyclopedia of Parallel Computing*. Ed. by D. A. Padua. Springer, 2011, pp. 1005–1006. ISBN: 978-0-387-09765-7.

[2] J. Dongarra and P. Luszczek. “ScaLAPACK”. In: *Encyclopedia of Parallel Computing*. Ed. by D. A. Padua. Springer, 2011, pp. 1773–1775. ISBN: 978-0-387-09765-7.

[3] J. J. Dongarra, P. Luszczek, and A. Petitet. “The LINPACK Benchmark: past, present and future”. In: *Concurrency and Computation: Practice and Experience* 15.9 (2003), pp. 803–820.

[4] R. van de Geijn and K. Goto. “Basic Linear Algebra Subprograms (BLAS)”. In: *Encyclopedia of Parallel Computing*. Ed. by D. Padua. Springer US, 2011, pp. 120–120. ISBN: 978-0-387-09765-7. DOI: 10.1007/978-0-387-09766-4_2066.

[5] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. “Basic Linear Algebra Subprograms for FORTRAN usage”. In: *ACM Trans. Math. Soft* 5 (1979), pp. 308–323.

[6] D. A. Padua, ed. *Encyclopedia of Parallel Computing*. Springer, 2011. ISBN: 978-0-387-09765-7.