MAX PLANCK
COMPUTING & DATA FACILITY

MPCDF

ScaLAPACK AND ELPA: HOW TO DIAGONALIZE REALLY LARGE DENSE MATRICES

Peter Karpov, Max Planck Computing and Data Facility, Garching

Summer School Paphos
October 3-8, 2023

OUTLINE

1. Introduction: eigensolver libraries
2. LAPACK
3. ScaLAPACK
4. ELPA
5. Showcases, benchmarks (if time permits)

MATRIX DIAGONALIZATION AND QUANTUM PROBLEMS

Matrix diagonalization is at the core of quantum problems:

$$\hat{H}|\psi\rangle = E|\psi\rangle$$

E_1, E_2, ... – eigenvalues (energy levels)
$|\psi_1\rangle$, $|\psi_2\rangle$, ... – eigenvectors (corresponding eigenstate wavefunctions)

Drawback of exact diagonalization:

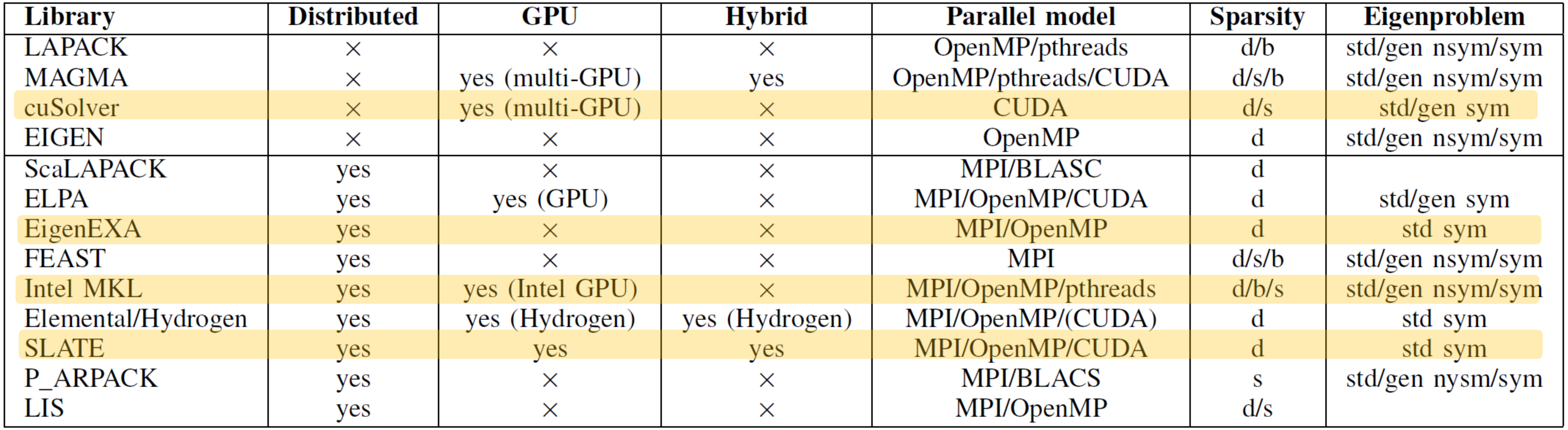
interacting particles	matrix size $\sim \exp(\# N_{\text{particles}})$
non-inteacting particles	matrix size $\sim N_{\text{particles}}$

Diagonalization is usually a bottleneck in DFT calculations!

DIAGONALIZATION METHODS

Direct methods	**Iterative methods**
⟶ all or substantial part (>10%) of eigenpairs	⟶ small part of eigenpairs (~several hundreds)
⟶ relatively small matrices (up to ~10^6)	⟶ much larger matrices (> 10^9)
⟶ dense matrices	⟶ (typically) sparse matrices
Software: LAPACK, ScaLAPACK, ELPA, …	Software: ARPACK, SLEPc, ChASE, …

PARALLEL DENSE EIGENSOLVERS



Library	Distributed	GPU	Hybrid	Parallel model	Sparsity	Eigenproblem
LAPACK	×	×	×	OpenMP/pthreads	d/b	std/gen nsym/sym
MAGMA	×	yes (multi-GPU)	yes	OpenMP/pthreads/CUDA	d/s/b	std/gen nsym/sym
cuSolver	×	yes (multi-GPU)	×	CUDA	d/s	std/gen sym
EIGEN	×	×	×	OpenMP	d	std/gen nsym/sym
ScaLAPACK	yes	×	×	MPI/BLASC	d	
ELPA	yes	yes (GPU)	×	MPI/OpenMP/CUDA	d	std/gen sym
EigenEXA	yes	×	×	MPI/OpenMP	d	std sym
FEAST	yes	×	×	MPI	d/s/b	std/gen nsym/sym
Intel MKL	yes	yes (Intel GPU)	×	MPI/OpenMP/pthreads	d/b/s	std/gen nsym/sym
Elemental/Hydrogen	yes	yes (Hydrogen)	yes (Hydrogen)	MPI/OpenMP/(CUDA)	d	std sym
SLATE	yes	yes	yes	MPI/OpenMP/CUDA	d	std sym
P_ARPACK	yes	×	×	MPI/BLACS	s	std/gen nysm/sym
LIS	yes	×	×	MPI/OpenMP	d/s	

ELPA Nov. 2022 world record:
diagonalization of **3.2M×3.2M** dense matrix

Davor Davidović, An overview of dense eigenvalue solvers for distributed memory systems, 44th International Convention on Information, Communication and Electronic Technology (2021)

OUTLINE

1. Introduction: eigensolver libraries
2. **LAPACK**
3. ScaLAPACK
4. ELPA
5. Showcases, benchmarks (if time permits)

LAPACK SOFTWARE FAMILY

	matrix-matrix operations	**“advanced” linear algebra**
sequential	BLAS	LAPACK
parallel	PBLAS	ScaLAPACK

BLAS Level 1 Routines:
vector-vector operations

BLAS Level 2 Routines:
matrix-vector operations

BLAS Level 3 Routines:
matrix-matrix operations
(e.g **matrix-matrix multiplication**)

systems of linear equations
linear least squares
eigenvalue problems
singular value decomposition

LAPACK / BLAS

$$\begin{bmatrix} L & A & P & A & C & K \\ L & -A & P & -A & C & -K \\ L & A & P & A & -C & -K \\ L & -A & P & -A & -C & K \\ L & A & -P & -A & C & K \\ L & -A & -P & A & C & -K \end{bmatrix}$$

Linear **A**lgebra **PACK**age
Basic **L**inear **A**lgebra **S**ubprograms

+ fast

+ extensively tested, stable

– cumbersome interface

Many packages use it under the hood:

NumPy/SciPy (python), Armadillo (C++), MATLAB, …

LAPACK / BLAS

No distributed memory parallelism

"**driver**" routines – complete solution

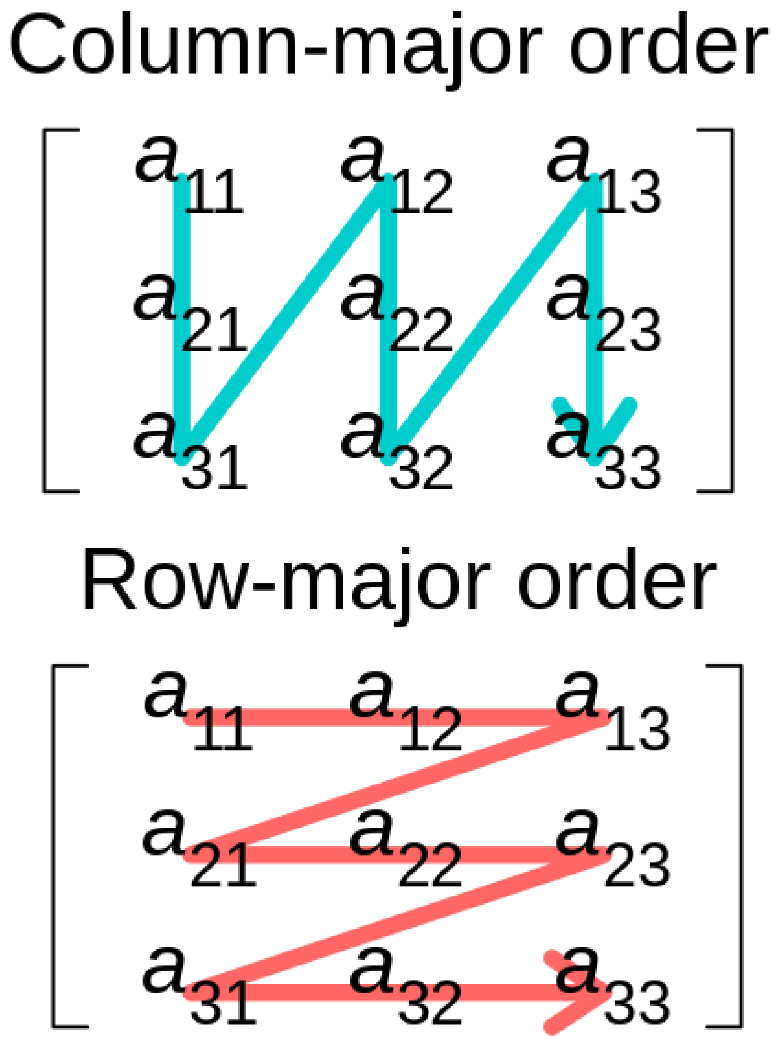
"**computational**" routines – complete one solution step

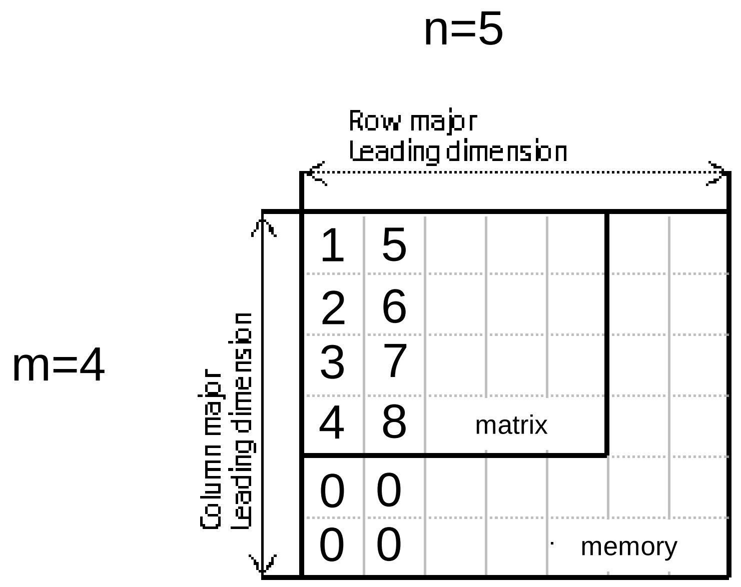
Naming convention: `pmmaa(a)`

`p` – precision:	`s,d`	–	real **s**ingle and **d**ouble precision
	`c,z`	–	complex single and double precision
`mm` – matrix type:	`sy`	–	**sy**mmetric
	`ge`	–	**ge**neral
`aa(a)` – algorithm:	`mm`	–	**m**atrix **m**ultiplication
	`evd`	–	**e**igen**v**alue problem with **d**ivide-and-conquer algorithm

Examples: `sgemm`, `dsyevd`

LAPACK: MATRIX REPRESENTATION


Column-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
Row-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}



Column-major memory representation:
A = {1, 2, 3, 4, 0, 0, 5, 6, 7, 8, 0, 0, …}

(column-major) leading dimension of matrix A:
lda = 6

EXAMPLE: MATRIX-MATRIX MULTIPLICATION WITH LAPACK

```
call dgemm (transa, transb, m, n, k,
     alpha, a, lda, b, ldb, beta, c, ldc)
```

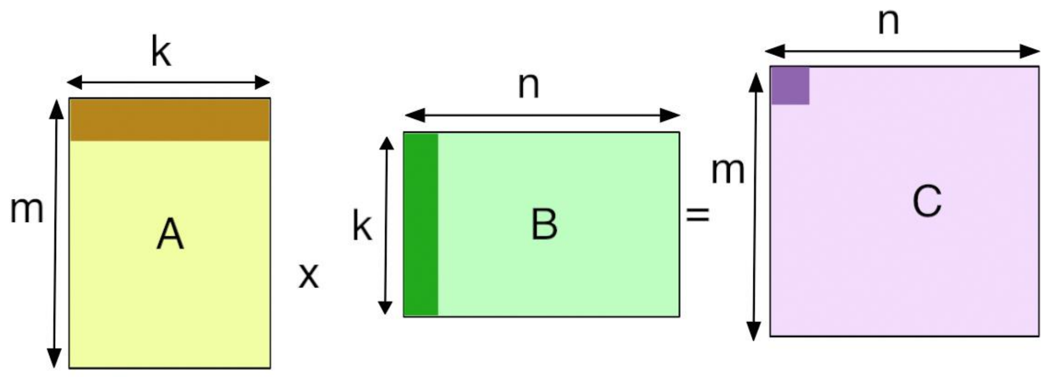


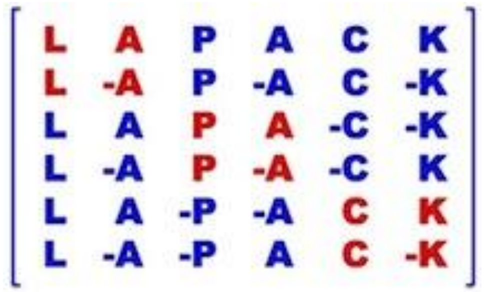
Figure: https://github.com/iVishalr/GEMM

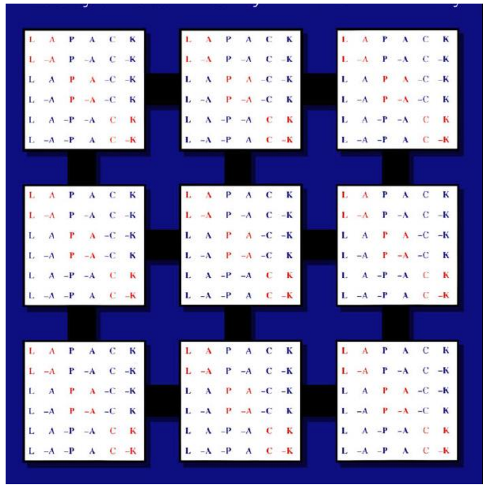
```
program dgemm_example
  implicit none
  double precision :: a(6), b(6), c(9)

  data a /1.d0, 1.d0, 2.d0, 2.d0, 3.d0, 3.d0/
  data b /1.d0, 2.d0, 3.d0, 1.d0, 2.d0, 3.d0/
  c = 0.d0

  call dgemm('N', 'N',
             3, 3, 2, 1.d0, a, 2, b, 3, 0.d0, c, 3)
end program
```

```
#include "cblas.h"

int main()
{
double a[6] = {1, 1, 2, 2, 3, 3};
double b[6] = {1, 2, 3, 4, 5, 6};
double c[9] = {0};

cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                 3, 3, 2, 1.0, a, 2, b, 3, 0.0, c, 3);

return 0;
}
```

OUTLINE

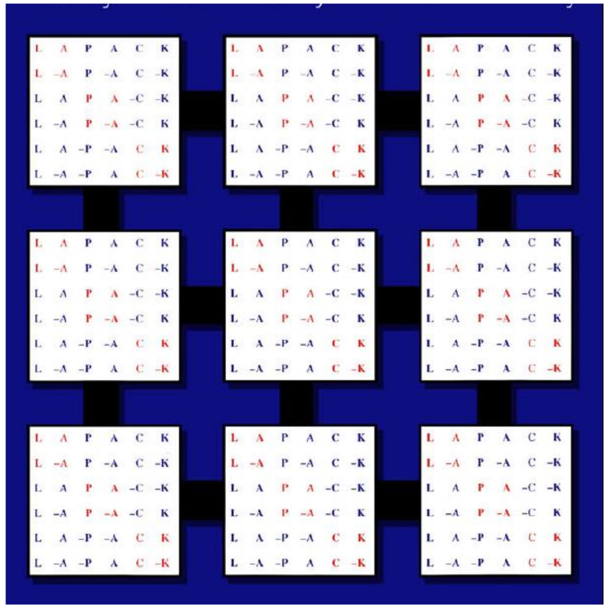
1. Introduction: eigensolver libraries
2. LAPACK
3. **ScaLAPACK**
4. ELPA
5. Showcases, benchmarks (if time permits)





ELPA

ScaLAPACK



Scalable **LAPACK** – MPI parallel version of LAPACK

Scaling: memory ~ N^2, time ~ N^3

Less functions (e.g. no eigensolver for non-symmetric matrices)

Naming: p+LAPACK name → ScaLAPACK name:

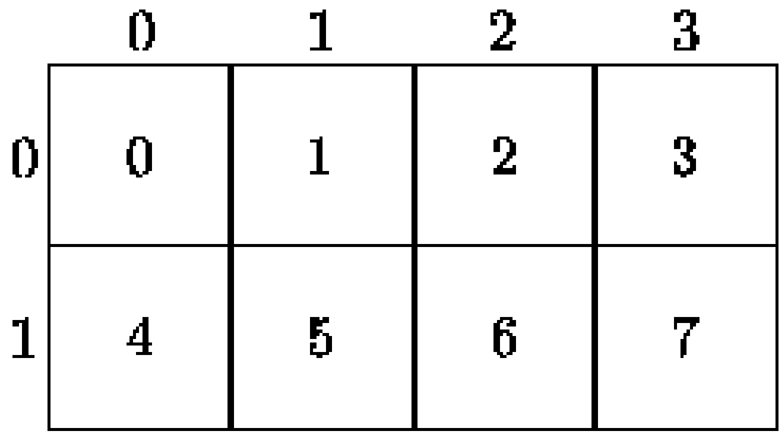
Examples: `sgemm` → `psgemm`

`dsyevd` → `pdsyevd`

Requires a special distribution of matrices among the processes:
"**block-cyclic matrix distribution**" → a major hurdle for the newcomers

ScaLAPACK: 2D PROCESS GRID

BLACS grid (Basic Linear Algebra Communication Subprograms)

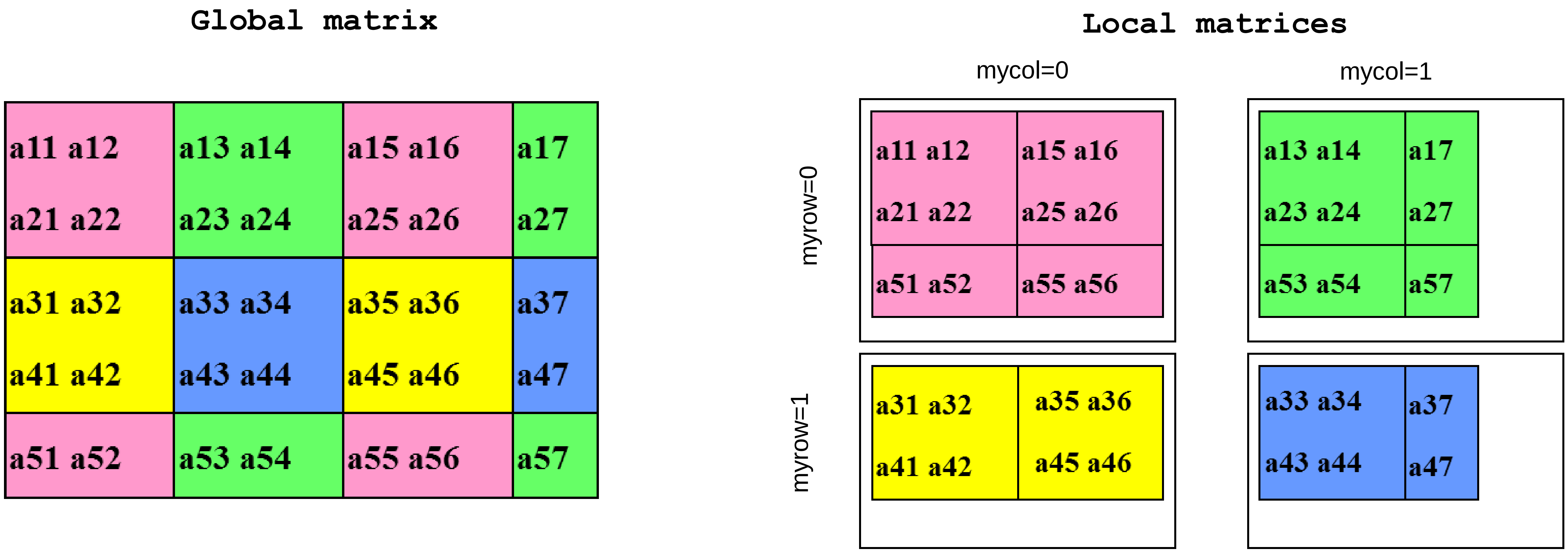


	0	1	2	3
0	0	1	2	3
1	4	5	6	7

MPI rank	BLACS grid coordinates
world_rank=0	myrow=0, mycol=0
world_rank=1	myrow=0, mycol=1
world_rank=2	myrow=0, mycol=2
world_rank=3	myrow=0, mycol=3
world_rank=4	myrow=1, mycol=0
…	…

ScaLAPACK: BLOCK-CYCLIC DISTRIBUTION

m,n	number of rows and columns of the matrix
mb, nb	sizes of blocks in columns and in rows
nprow, npcol	number of rows and columns of the two dimensional process grid
rsrc, csrc	row and column index of the process containing the first element of the matrix



m=5,n=7, mb=nb=2, nprow=npcol=2, rsrc=csrc=0

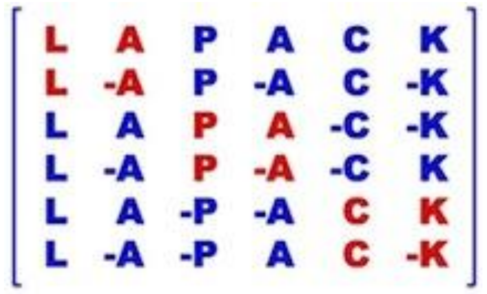
https://info.gwdg.de/wiki/doku.php?id=wiki:hpc:scalapack

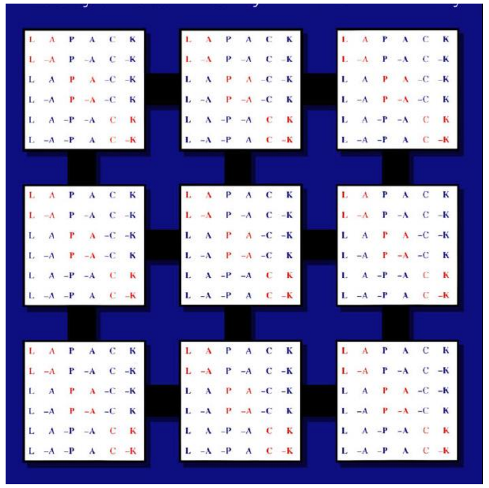
EXAMPLE: FIND ALL EIGENVALUES AND EIGENVECTORS OF A SYMMETRIC MATRIX WITH SCALAPACK

```
! Initialize MPI
call MPI_Init(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, world_size, ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, world_rank, ierr)

! BLACS grid initialization
call blacs_get(0, 0, ictxt)
call blacs_gridinit(ictxt, 'row', nprow, npcol)
call blacs_gridinfo(ictxt, nprow, npcol, myrow, mycol)

! Compute local matrix size
call numroc(m_loc, N, myrow, 0, nprow, NB)
call numroc(n_loc, N, mycol, 0, npcol, NB)

! Initialize array descriptors for distributed matrix A and Z
call descinit(descA, N, N, NB, NB, 0, 0, ictxt, m_loc, info)
call descinit(descZ, N, N, NB, NB, 0, 0, ictxt, m_loc, info)

  ! Allocate local storage for distributed matrix A and Z (eigenvector matrix)
allocate(A_loc(m_loc, n_loc))
allocate(Z_loc(m_loc, n_loc))
! Allocate space for eigenvalues -- global array
allocate(Eigenvalues(N))

call pdsyev('V', 'U', N, A_loc, ione, ione, descA, Eigenvalues, Z_loc, ione, ione, descZ, WORK, LWORK, info)
! "N" - only eigenvalues, "V" - eigenvalues and eigenvectors; A-??
! "U" - use upper,  "L" - lower triangular matrix (its symmetric anyhow)
```

OUTLINE

1. Introduction: eigensolver libraries
2. LAPACK
3. ScaLAPACK
4. **ELPA**
5. Showcases, benchmarks (if time permits)





ELPA

now we target **Exaflop** and beyond

Eigenvalue soLvers for Petaflop Applications (started at 2008 at MPCDF)
(Eigenwert-Löser für Petaflop-Anwendungen)

Direct eigensolver for **dense** large-scale symmetric/hermitian matrices

ELPA vs ScaLAPACK:

- ELPA is eigensolver, not general purpose linear algebra

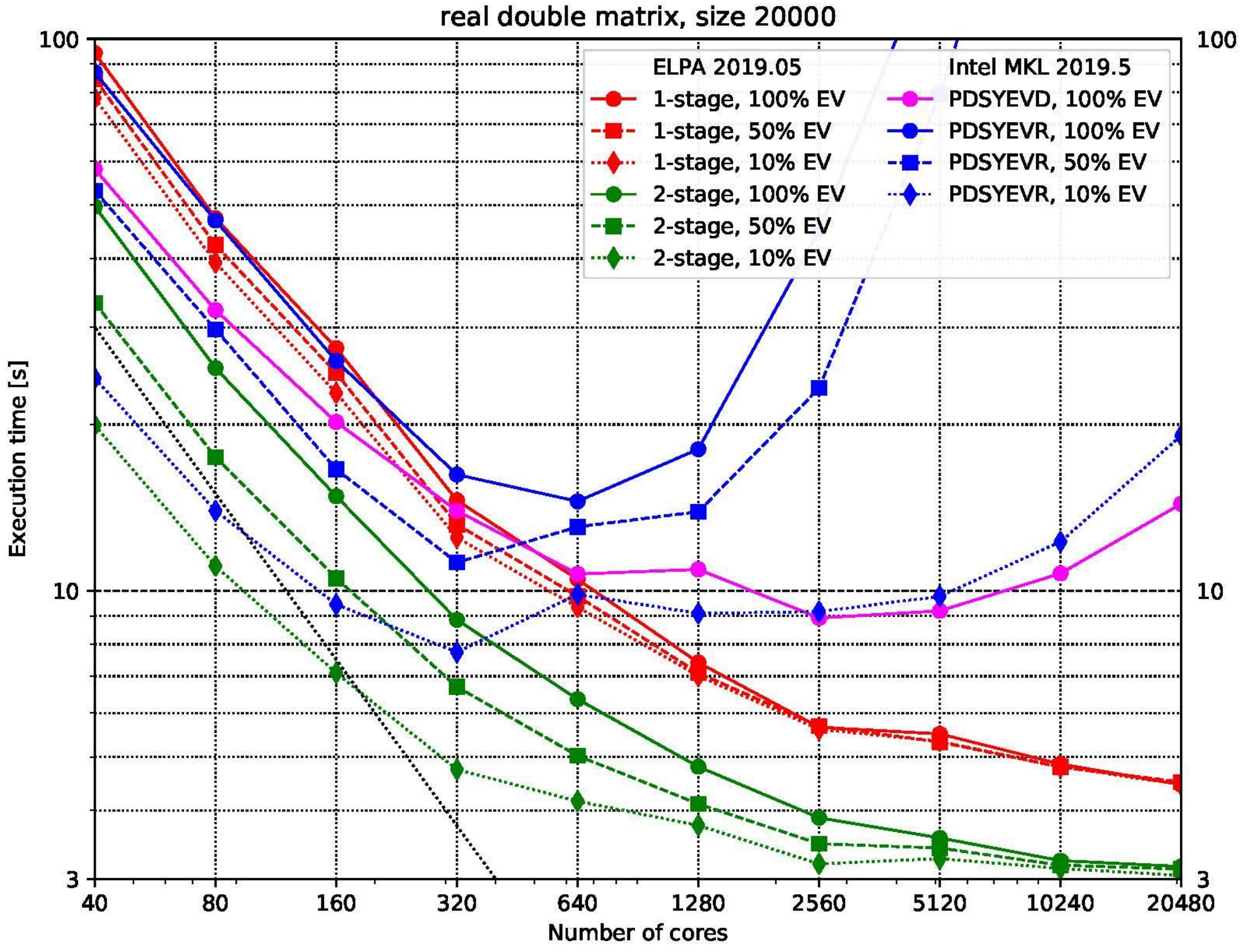
- ELPA is up to ~x2 faster than ScaLAPACK

- **ELPA works on GPUs** (NVIDIA, AMD, Intel)

- ELPA uses same "block-cyclic" matrix layout as ScaLAPACK

- ABINIT
- BerkeleyGW
- CP2K
- CPMD
- DFTB+
- EIGENKERNEL
- ELSI
- FHI-aims
- GPAW
- NWChem
- Octopus
- OpenMM
- OpenMX
- QuantumATK
- QuantumEspresso
- SIESTA
- VASP
- WIEN2k

ELPA VS ScaLAPACK

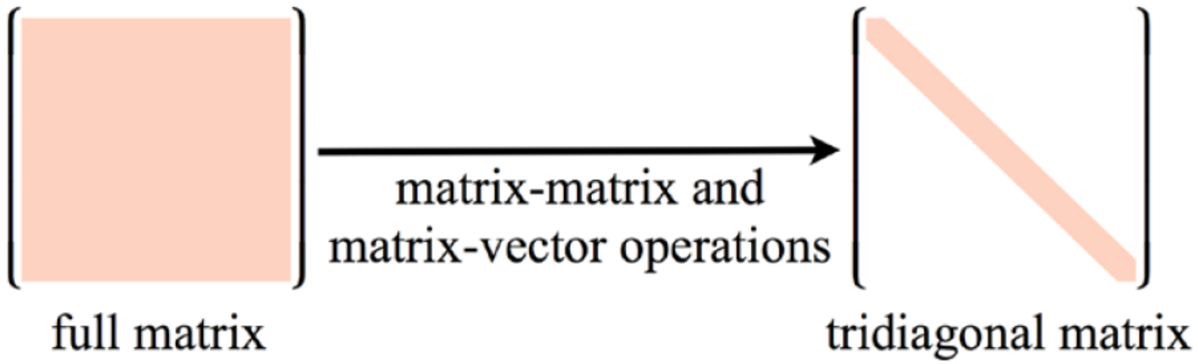


ELPA’s additional perks:

- Generalized eigenproblem
- Antisymmetric eigenproblem
- PDGEMM-GPU (coming soon!)

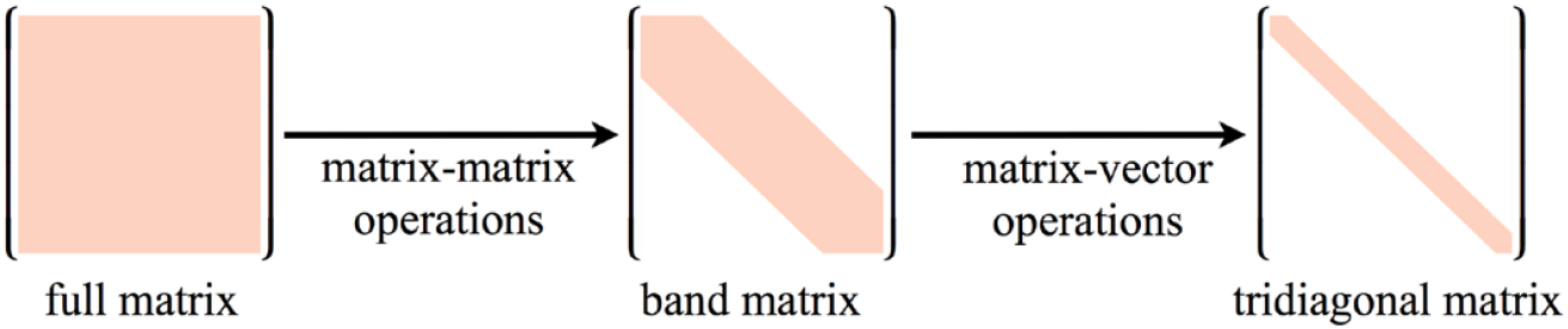
ELPA: 1- AND 2-STAGE SOLVERS

ELPA1 (one stage solver)
- for GPUs
- for the whole eigenspectrum



ELPA2 (two stage solver)
- for CPU
- for part of eigenspectrum

idea: Bruno Lang

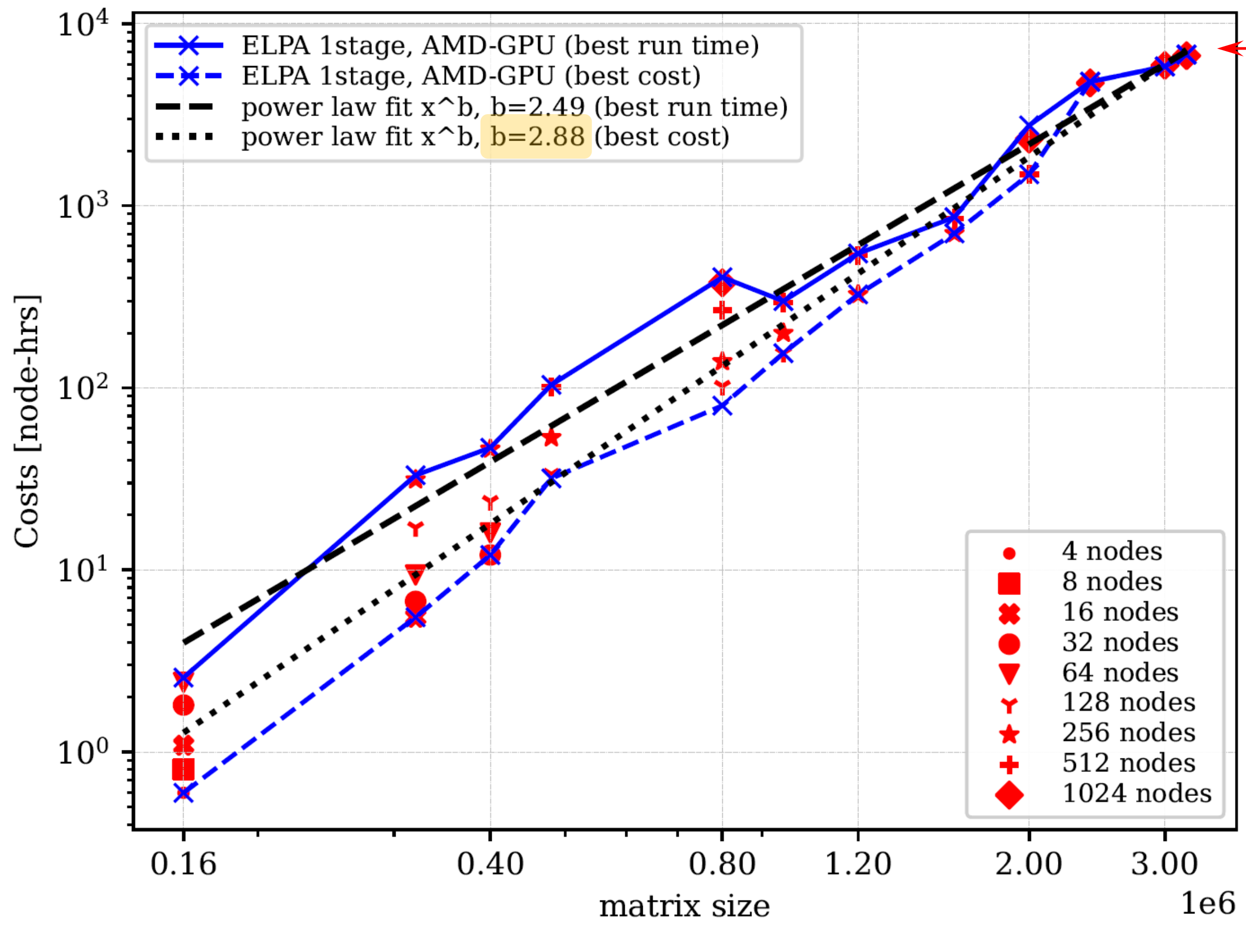


A. Marek, V. Blum et al, J. Phys.: Condens. Matter 26 213201 (2014)

OUTLINE

1. Introduction: eigensolver libraries
2. LAPACK
3. ScaLAPACK
4. ELPA
5. **Showcases, benchmarks** (if time permits)

FULL SUPPORT FOR AMD GPU'S: READY TO USE! (since 2022.11)



Nov. 2022 world record
3.2 M × 3.2 M matrix
(1000 nodes, 4000 GPUs, ELPA1)

Benchmarks on pre-exascale LUMI, Finland
(2500 nodes, 4 AMD Mi250x GPUs)

RESOURCES

LAPACK user guide	https://www.netlib.org/lapack/lug/
ScaLAPACK user guide	https://www.netlib.org/scalapack/slug/
Intel MKL	Developer Reference for Intel® oneAPI Math Kernel Library for C Developer Reference for Intel® oneAPI Math Kernel Library for Fortran
Short intro from GWDG	https://info.gwdg.de/wiki/doku.php?id=wiki:hpc:scalapack

ELPA user guide (preview)

Tutorials today: "HPC"

Need help with ELPA? petr.karpov@mpcdf.mpg.de