Advancing the distributed Multi-GPU ChASE library through algorithm optimization and NCCL library

Xinzhe Wu and **Edoardo Di Napoli**

Jülich Supercomputing Centre

Germany

EIGELSOLVER LIBRARIES

Complexity

Problem definition

$$AX = X\Lambda \quad A \equiv A^H \in \mathbb{C}^{n\times n} \quad X \in \mathbb{C}^{n\times k} \quad \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^{k\times k} \quad k < n$$

EIGELSOLVER LIBRARIES

Complexity

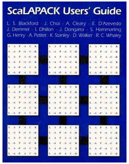
Problem definition

$$AX = X\Lambda \quad A \equiv A^H \in \mathbb{C}^{n\times n} \quad X \in \mathbb{C}^{n\times k} \quad \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^{k\times k} \quad k < n$$

Eigensolver algorithms based on direct diagonalization (dense matrices)

- Divide&Conquer
- MRRR
- BXInvIt
- ...

$$\mathcal{O}(n^3)$$



ELPA

EIGELSOLVER LIBRARIES

Complexity

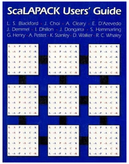
Problem definition

$$AX = X\Lambda \quad A \equiv A^H \in \mathbb{C}^{n\times n} \quad X \in \mathbb{C}^{n\times k} \quad \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^{k\times k} \quad k < n$$

Eigensolver algorithms based on direct diagonalization (dense matrices)

- Divide&Conquer
- MRRR
- BXInvIt
- ...

$$\mathcal{O}(n^3)$$



Eigensolver based on iterative algorithms (sparse matrices)

- Subspace iteration
- Krylov methods
- Rayleigh-Ritz projection (e.g. LOBPCG)
- ...

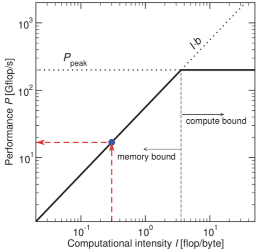
$$\mathcal{O}(k \times n^2)$$

EIGELSOLVER LIBRARIES

Guiding principles for performance and scaling

Given an algorithm . . .

1. Blocked algorithms to maximize **computational intensity**.
2. Avoid as much as possible to **communicate** data across computing units or processes.
3. Even when communication is unavoidable, maximize **memory bandwidth** usage.

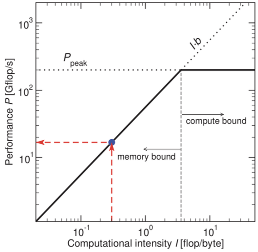


EIGELSOLVER LIBRARIES

Guiding principles for performance and scaling

Given an algorithm . . .

1. Blocked algorithms to maximize **computational intensity**.
2. Avoid as much as possible to **communicate** data across computing units or processes.
3. Even when communication is unavoidable, maximize **memory bandwidth** usage.



Overall algorithm is $\mathcal{O}(k \times n^2)$, some kernels could have **lower complexity**

One more guiding principle

4. If possible, use for each kernel the appropriate **level** of parallelism.

$\Longrightarrow$ **Subspace iteration powered by spectral filters**

A KNOWLEDGE-INCLUSIVE OPTIMIZED EIGENSOLVER

ChASE
CHEBYSHEV ACCELERATED SUBSPACE EIGENSOLVER

- License: open source — BSD 3.0
- GitHub: `https://github.com/ChASE-library/ChASE`
- Docs: `https://chase-library.github.io/ChASE/index.html`
- Latest release: v. 1.4.0 – August 7th 2023
- Zenodo Key: `https://doi.org/10.5281/zenodo.6366000`
- Reference key: `https://doi.org/10.1145/3313828`
- Reference key: `https://doi.org/10.1145/3539781.3539792`

Highlights

- Solve for Symmetric real/Hermitian complex eigenproblems
- Sequences of dense eigenproblems: exploits correlation between adjacent problems
- Modern C++ interface: depends only on LAPACK and BLAS functions
- Distributed CPU and multi-GPU builds available
- Easy-to-integrate: ready-to-use Fortran to C++ interface

USE CASES AND FEATURES

- ChASE is templated for Real and Complex type.
- ChASE is also templated to work in Single and Double precision.
- ChASE is currently designed to solve for the extremal portion of the eigenspectrum. The library is particularly efficient when **no more than 20%** of the eigenspectrum is sought after.
- ChASE currently handles standard eigenvalue problems.
- ChASE can receive as input a matrix of vector $\hat{V}$
- For a fixed accuracy level (residual tolerance), ChASE can optimize the degree of the Chebyshev polynomial filter so as to minimize the number of FLOPs necessary to reach convergence.

CHEBYSHEV SUBSPACE ITERATION ALGORITHM

v1.2.2

INPUT: Hermitian matrix A, `tol`, `deg` — OPTIONAL: approximate eigenvectors V, extreme eigenvalues $\{\lambda_1, \lambda_{\text{NEV}}, \lambda_{\text{MAX}}\}$.

OUTPUT: NEV wanted eigenpairs (Λ, V).

1. **Lanczos DoS step.** Identify the bounds for $\{\lambda_1, \lambda_{\text{NEV}}, \lambda_{\text{MAX}}\}$ corresponding to the wanted eigenspace.

REPEAT UNTIL CONVERGENCE:

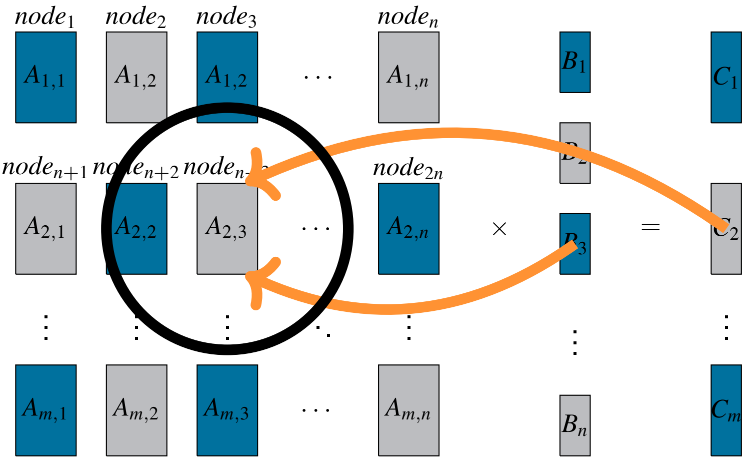
2. **Optimized Chebyshev filter.** Filter a block of vectors $V \longleftarrow p(A)V$ with optimal degree.
3. Re-orthogonalize the vectors outputted by the filter; $V = QR$.
4. Compute the Rayleigh quotient $G = Q^\dagger AQ$.
5. Compute the primitive Ritz pairs (Λ, Y) by solving for $GY = Y\Lambda$.
6. Compute the approximate Ritz pairs $(\Lambda, V \leftarrow QY)$.
7. Compute the residuals of the Ritz vectors $\|AV - V\Lambda\|$.
8. Deflate and lock the converged vectors.

END REPEAT

Legend: Original algorithmic contribution, 2D MPI parallel, executed redundantly on each process

MATRIX AND VECTORS DISTRIBUTION

- Each node gets the appropriate part of A, B and C.



ENVIRONMENT AND EIGENPROBLEM TYPE

JURECA-DC GPU partition

- 2×64 cores AMD EPYC 7742 CPUs @ 2.25 GHz (16×32 GB DDR4 Memory)
- 4 NVIDIA Tesla A100 GPUs (4×40 GB high-bandwidth memory).
- ChASE (relase 1.1.2) is compiled with GCC 9.3.0, OpenMPI 4.1.0 (UCX 1.9.0), CUDA 11.0 and Intel MKL 2020.4.304.
- All computations are performed in double-precision.

Table: Spectral information for generating test matrices. In this table, we have $k = 1, \cdots, n$.

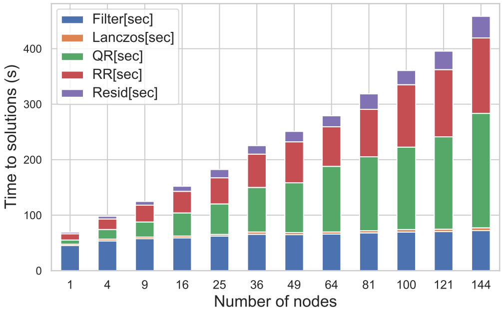
Matrix Name	Spectral Distribution
UNIFORM (UNI)	$\lambda_k = d_{max}(\epsilon + \frac{(k-1)(1-\epsilon)}{n-1})$
GEOMETRIC (GEO)	$\lambda_k = d_{max}\epsilon^{\frac{n-k}{n-1}}$
(1-2-1) (1-2-1)	$\lambda_k = 2 - 2\cos(\frac{\pi k}{n+1})$
WILKINSON (WILK)	All positive, but one, roughly in pairs.

PASC22 proceedings: `https://doi.org/10.1145/3539781.3539792`

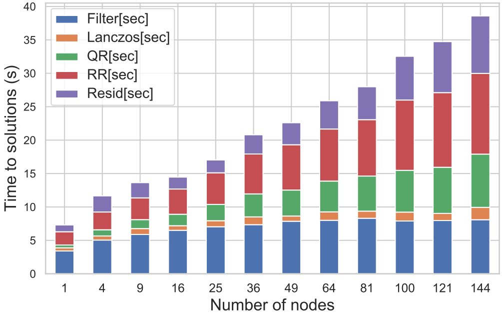
WEAK SCALING

Artificial matrices: type UNIFORM**, from** $n = 30000$ **until** $n = 360000$**, nev** $= 2250$ **and nex** $= 750$

CPU scaling



GPU scaling



- $4 \times$ GPUs with 1 MPI task per node;
- ChASE scales linearly;
- Time doubles every time matrix size quadruples (CPU) and triples (GPU);
- Filters scales very well;
- Confirm `QR, RR, Resid` need a revised parallel computational scheme.

NEW PARALLEL ALGORITHM

for QR, Rayleigh-Ritz and Residuals

Chase Algorithm

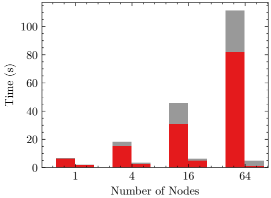
- Changed workspace design $\Longrightarrow$ reduction in memory consumption
- 1-D distribution for array of vectors in QR factorization, Rayleigh-Ritz (RR) projection, and Residual computation
- Hiding communication with computation within for RR projection and Residual computation
- Hybrid usage of Householder- and Cholesky-QR for the QR factorization
- Mechanism to limit polynomial degree to avoid the failure of CholQR
- New release: Version v1.3.0 (March 10th 2023)
- **Much better strong and weak scaling**

1D-MPI VS REDUNDANT ON EACH MPI

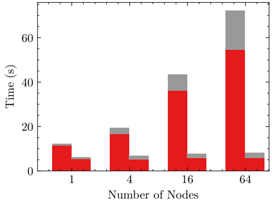
1^{st} row: JURECA-DC (1 interconnect) – 2^{nd} row: JUWELS Booster (4 interconnects)

WS: Artificial matrices: type UNIFORM, from $n = 30000$ until $n = 240000$, nev $= 2250$ and nex $= 750$

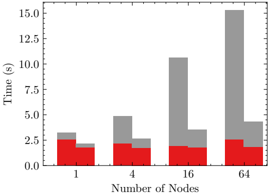
QR CPU



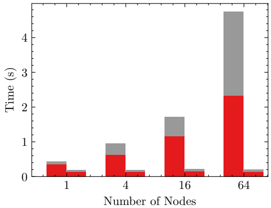
RR CPU



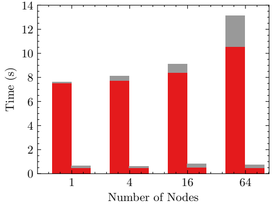
Resid CPU



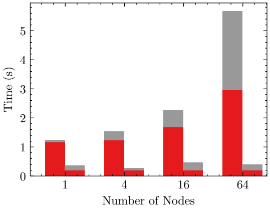
QR GPU



RR GPU



Resid GPU



Computation (red) and communication (gray)

ChASE v1.2.1 (solid color) and ChASE v1.3.0 (hatch style color)

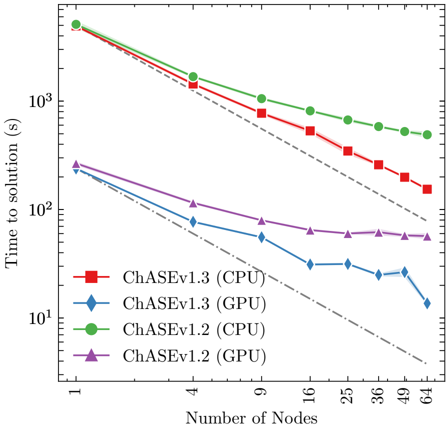
JÜLICH Forschungszentrum

WEAK AND STRONG SCALING

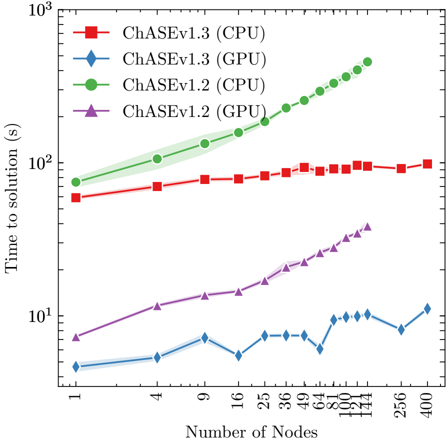
SS: Artificial matrix: type UNIFORM, $n = 130000$, nev $= 1000$ and nex $= 300$

WS: Artificial matrices: type UNIFORM, from $n = 30000$ until $n = 600000$, nev $= 2250$ and nex $= 750$

Strong scaling



Weak scaling



JÜLICH Forschungszentrum

EXPLOITING NCCL

Memory copying operations for the collective operations can be bypassed by exploring the GPUDirect technology

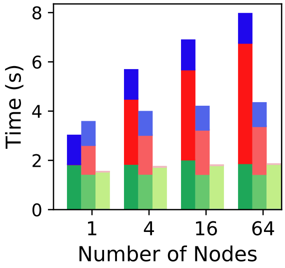
- Used GPU-driven NCCL library to replace the MPI library for all the collective communication;
- 2D NCCL communicator has been built on top of the 2D MPI grid;
- Each MPI process is mapped to a single GPU device;
- All the operations of AllReduce and Bcast are substituted by their equivalents in NCCL;
- All the host-device data movement for all major kernels have been eliminated.

NCCL VS 1D-MPI VS REDUNDANT ON EACH MPI

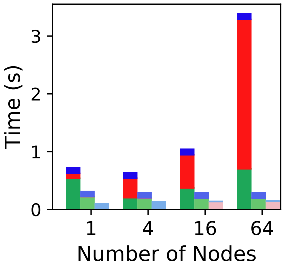
JUWELS Booster (4 interconnects)

WS: Artificial matrices: type UNIFORM, from $n = 30000$ until $n = 240000$, nev $= 2250$ and nex $= 750$

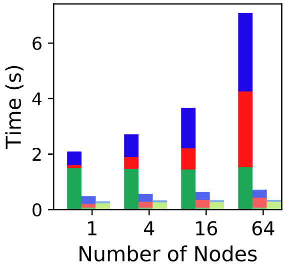
Filter GPU



Resid GPU



RR GPU



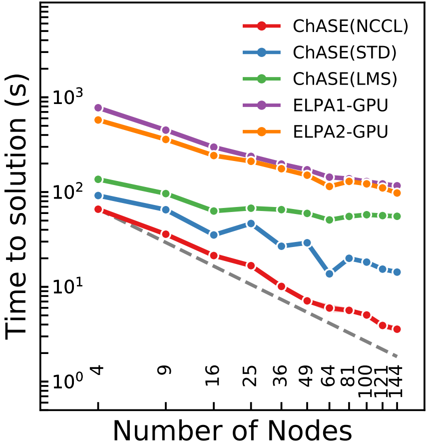
Computation (marked in green), communication (red) and data movement (blue)
ChASE LMS (v1.2.2) — bright color shades
ChASE STD (v1.3.0) — lighter color shades
ChASE NCCL (v1.4.0) — lightest color shades

WEAK AND STRONG SCALING

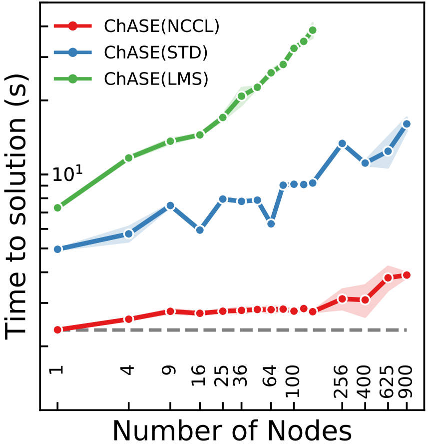
SS: MS matrix: In_2O_3, $n = 115000$, nev $= 1200$ and nex $= 400$

WS: Artificial matrices: type UNIFORM, from $n = 30000$ until $n = 900000$, nev $= 2250$ and nex $= 750$

Strong scaling



Weak scaling



JÜLICH Forschungszentrum

LESSONS LEARNED

1. Design of kernel parallelism has to evolve with the size of the problem;
2. Strategy to avoid communication had to evolve with the evolution of the hardware;
3. Be on the lookout to exploit new algorithms (CholQR)
4. Extracting node-level performance using specialized kernels is not trivial;
5. Avoiding communication may come at the cost of increasing memory usage an decreased parallelism $\longrightarrow$ need to strike a careful trade-off (new 1D parallelization of some kernels);
6. Initialization can become a substantial bottleneck for large scale computations.

OUTLOOK

- Porting to FUGAKU on the way (aim: learn some lessons towards Jupiter exascale modular booster)
- Next bottleneck: solving for $n \sim \mathcal{O}(10^6)$ and nev$> 0.001 \times n \rightarrow$ mixed 2D distribution (block-cyclic + element-wise)
- Extension to interior eigenproblems through rational spectral filters for sparse matrices $n \sim \mathcal{O}(10^7 - 10^8)$ with flexible 3D distribution
- (Adaptive) integration in domain software (FHI-aims, QE);
- Explore extension to mixed-precision.

OUTLOOK

- Porting to FUGAKU on the way (aim: learn some lessons towards Jupiter exascale modular booster)
- Next bottleneck: solving for $n \sim \mathcal{O}(10^6)$ and nev$> 0.001 \times n \rightarrow$ mixed 2D distribution (block-cyclic + element-wise)
- Extension to interior eigenproblems through rational spectral filters for sparse matrices $n \sim \mathcal{O}(10^7 - 10^8)$ with flexible 3D distribution
- (Adaptive) integration in domain software (FHI-aims, QE);
- Explore extension to mixed-precision.

Thank you!

References

Berljafa, Wortmann, Di Napoli – https://10.1002/cpe.3394 (2014)

Winkelmann, Springer, Di Napoli – https://doi.org/10.1145/3313828 (2019)

Zhang, Achilles, Winkelmann, Haas, Schleife, Di Napoli – https://doi.org/10.1016/j.cpc.2021.108081 (2021)

Wu, Davidovic, Achilles, Di Napoli – https://doi.org/10.1145/3539781.3539792 (2022)

Wu, Di Napoli – https://doi.org/10.1145/3624062.3624249 (2023)

CHASE LIBRARY

MPI configurations

- Shared memory build: to be used on only one computing node or on a single CPU
- MPI + X build: to be used on multi-core homogeneous CPU clusters
- GPU build: to be used on heterogeneous computing clusters. Currently we support the use of one or more GPU cards per computing node in a number of flexible configurations

Usage

- Free standing library compiles with CMake
- Used a submodule by linking the library

Parallel distribution

- Custom 2D block distribution
- 2D block-cyclic distribution (a-la-ScaLAPACK)

Fortran and C interfaces

- Integrated in `devel` version of FLEUR (CPU only)
- Integrated in `devel` version of Quantum ESPRESSO (CPU only)
- Integrated in release version of Jena BSE code (both CPU and GPU version)
- Integrated in ELSI (ChASEv1.2.2)