ChASE - A Distributed Hybrid CPU-GPU Eigensolver for Large-scale Hermitian Eigenvalue Problems

Xinzhe Wu[2], Davor Davidović[1], Sebastian Achilles[2], Edoardo Di Napoli[2]

[1]Centre for Informatics and Computing, Ruđer Bošković Institute, Zagreb, Croatia

[2]Jülich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany

PASC22 Conference,
27 - 29 June 2022, Basel, Switzerland

ChASE
CHEBYSHEV ACCELERATED SUBSPACE EIGENSOLVER

Motivation I

- The dense Hermitian eigenproblems are their core computational problem in numerous research problems $\rightarrow$ e.g. condensed matter physics
- Computing eigenvectors and eigenvalues for extremely large matrices is still a significant computational challenge
- There are only a few existing implementations capable of utilizing the distributed heterogeneous (GPU-based) systems
- But, up to our knowledge no library is able to successfully exploit GPUs for dense eigenproblems with size larger than 100000

The main challenges

The **communication** overhead and how to exploit the full potential of the **heterogeneous multi-GPU** based systems

Motivation I

- The dense Hermitian eigenproblems are their core computational problem in numerous research problems $\rightarrow$ e.g. condensed matter physics
- Computing eigenvectors and eigenvalues for extremely large matrices is still a significant computational challenge
- There are only a few existing implementations capable of utilizing the distributed heterogeneous (GPU-based) systems
- But, up to our knowledge no library is able to successfully exploit GPUs for dense eigenproblems with size larger than 100000

The main challenges

The **communication** overhead and how to exploit the full potential of the **heterogeneous multi-GPU** based systems

Table of contents

Outline

Problem formulation

Eigenvalue problem

Partial diagonalization of *nev* eigenpairs of standard symmetric eigenvalue problem:

$A\ V = \Lambda\ V$

A is a $n \times n$ symmetric/Hermitian matrix, V is a $n \times nev$ rectangular and Λ is a $nev \times nev$ diagonal matrix.

The problems we tackle

- A is symmetric or Hermitian
- Only a small fraction of extremal eigenpairs are sought-after
- Large-scale eigenproblems ($O(100000)$)

State-of-the-art solutions

- **Direct solvers**: reduce matrix to condensed form (tridiagonal/band) and then computes eigenpairs (MRRR, D&C, QR) $\rightarrow$ complexity $O(n^3)$
- **Iterative solvers**: project the eigenproblem on a smaller, gradually improved search space. Approximate eigenpairs from the search space.

Existing solutions

- Iterative methods have proved to be a better choice if only a subset of eigenpairs are required!
- Direct solvers (libraries): ScaLAPACK, EigenEXA, ELPA
- Iterative solvers: FEAST and PFEAST (for dense Hermitian problems)
- (up to our knowledge) only ELPA supports the execution on the distributed GPUs
- But there are numerous shared-memory solutions: MAGMA, cuSOLVER(-MG), etc.

Outline

ChASE library

- ChASE is open source (BSD 2.0 licences)
- GitHub: `https://github.com/ChASE-library/ChASE`
- `https://doi.org/10.1145/3313828`

Highlights

- Chebyshev polynomial with degree optimization to accelerate convergence
- Accurately approximates the extremal eigenvalues of dense Hermitian eigenproblems
- Particularly effective on solving a sequence of correlated eigenproblems
- Modern C++ interface: easy-to-integrate in application codes

ChASE algorithm - pseudocode

1: Set the initial vector of degrees $m_{1:\text{nev+nex}} \leftarrow deg$
2: **Lanczos** step: Compute spectral estimates μ_1 and $\mu_{\text{nev+nex}}$ for lower/upper bound of eigenspectrum and $b_{sup} > \lambda_n$
3: Set initial estimate vectors $\hat{V}$
4: **while** size($\hat{Y} <$ nev) **do**
5: **Chebyshev filter**: Filter a block of vectors $\hat{V}$ with a vector of degree m
6: **Re-orthogonalize** vectors as $Q \leftarrow \texttt{QR}([\hat{Y}\ \hat{V}])$
7: Compute the **Rayleigh** quotient: $G = Q^* A Q$
8: Compute primitive **Ritz pairs**: $(\hat{V}, \hat{\Lambda})$ by solving $GZ = \Lambda Z$
9: Compute **approximate Ritz pairs**: $\hat{V} \leftarrow QZ$
10: Compute **residuals** of eigenpairs $Res(\hat{V}, \tilde{\Lambda})$
11: **Deflate** and **Lock** the converged eigenvectors and values from $(\hat{V}, \hat{\Lambda})$
12: **Optimize** the vector of polynomial degrees m
13: **end while**

ChASE algorithm

- Can be cast almost entirely in terms of BLAS-3 operations → easily exploits optimized BLAS and LAPACK libraries (e.g. MKL, OpenBLAS, libFLAME)
- Existing ChASE take advantage of optimized BLAS/LAPACK libraries for `QR`, `Rayleigh-Ritz`
- Except for the most time consuming kernel - Hermitian matrix-matrix multiplication used in:
  - `Filter`, `Rayleigh-Ritz` and `Residual`
  - Customized MPI scheme implementation

Parallel implementation

- Matrix A subdivided into 2D (square) grid of blocks, each one assigned to one MPI processes in 2D block fashion
- Block of vectors $\hat{V}$ are split in a 1D block fashion and distributed among the row communicators
- Custom HEMM implementation for a 2D grid distribution of A

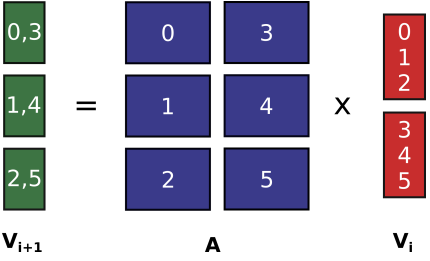
$$A_{dist} = \left(\begin{array}{c|c} A_{0,0} & A_{0,1} \\ \hline A_{1,0} & A_{1,1} \\ \hline A_{2,0} & A_{2,1} \end{array}\right), \; \hat{V}_{dist} = \left(\begin{array}{c|c} \hat{V}_0 & \hat{V}_1 \\ \hline \hat{V}_0 & \hat{V}_1 \\ \hline \hat{V}_0 & \hat{V}_1 \end{array}\right), \; \hat{W}_{dist} = \left(\begin{array}{c|c} \hat{W}_0 & \hat{W}_0 \\ \hline \hat{W}_1 & \hat{W}_1 \\ \hline \hat{W}_2 & \hat{W}_2 \end{array}\right),$$

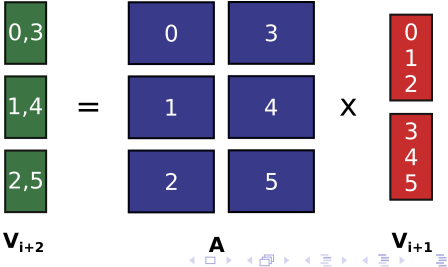
Chebyshev Filter

- In the filter the Hermitian matrix-matrix multiplication appears as a three-terms recurrence relation:

$$\hat{V}_{i+1} = \alpha_i (A - \gamma_i I_n) \hat{V}_i + \beta_i \hat{V}_{i-1}, \quad i \in [1, m), \tag{1}$$

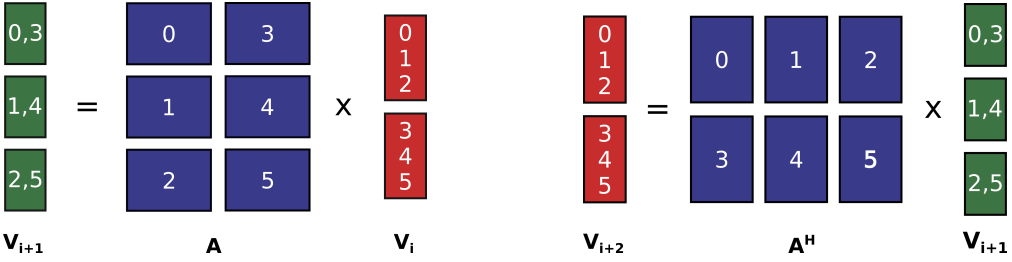
m is the degree of Chebyshev polynomial, $\alpha_i, \beta_i, \gamma_i$ are scalar parameters of each iteration i





Chebyshev filter

- The V_i is required to be redistributed between iteration i and $i+1 \rightarrow$ additional communication and memory copy
- To avoid redistribution one can right-multiply with A^H



Outline

Distributed ChASE

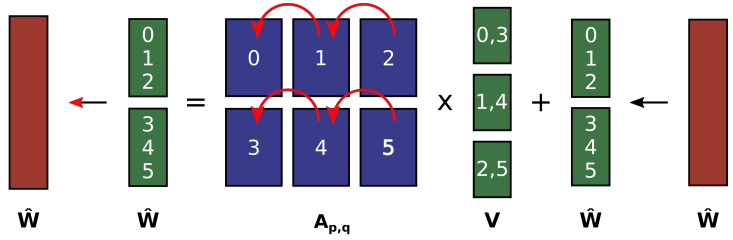
Current state

- ChASE only offloads the HEMM operator to GPU (single GPU per MPI rank per compute node)
- Other routies are parallelized using optimized parallel BLAS/LAPACK routines

Multi-GPU ChASE

- Use existing 2D block division and MPI 2D grid
- Per MPI rank subdivide into 2D grid of blocks and statically distributed among GPUs
- The sub-blocks of A are copied to and remain in the GPU memory until ChASE completes - avoids continuous memory copies to/from the GPU memory

Distributed ChASE II



Distributed ChASE III

- Matrix A should fit in the aggregated memory of all available GPUs
- Execution between the GPUs only within the MPI rank (e.g. GPUs on the same node)
- All other linear algebra routines (except `HEMM`) are computed redundantly on each MPI process
  - The most time consuming kernels are offloaded to GPUs using cuBLAS and cuSolver libraries (smaller matrix multiplications, QR)

Memory requirements

Per MPI process memory requirements

$$M_{cpu} = pq + (p+q)n_e + 2n_e n,$$

n is the rank of matrix A, $n_e = \text{nev} + \text{nex}$ is the largest dimension of the active subspace, 2D MPI grid is defined as $r \times c$, the local matrix held by each MPI rank is $p \times q$, where $p = \frac{n}{r}$ and $q = \frac{n}{c}$.

Per GPU memory requirement

$$M_{gpu} = \frac{pq}{r_g c_g} + 3\max(\frac{p}{r_g}, \frac{q}{c_g})n_e + (2n + n_e)n_e,$$

multiple GPUs can bind to an MPI process as a $r_g \times c_g$ 2D grid scheme.

Outline

Test suite

Compute infrastructure

- JURECA-DC supercomputer at Jülich Supercomputing Centre,
- Node: two 64 cores AMD EPYC 7742 CPUs @ 2.25 GHz (512 GB DDR4 Memory) and four NVIDIA Tesla A100 GPUs (4×40 GB high-bandwidth memory)

Test matrices

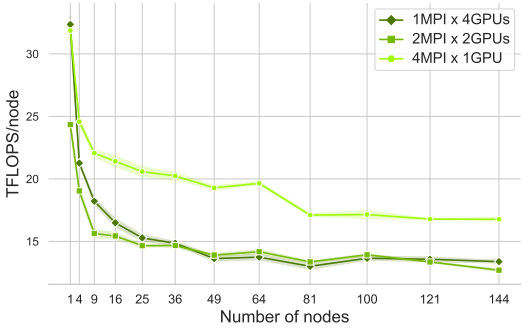
Matrix Name	Spectral Distribution
Uniform (Uni)	$\lambda_k = d_{max}(\epsilon + \frac{(k-1)(1-\epsilon)}{n-1})$
Geometric (Geo)	$\lambda_k = d_{max}\epsilon^{\frac{n-k}{n-1}}$
(1-2-1) (1-2-1)	$\lambda_k = 2 - 2\cos(\frac{\pi k}{n+1})$
Wilkinson (Wilk)	All positive, but one, roughly in pairs.

All tests are performed in double precision arithmetic.

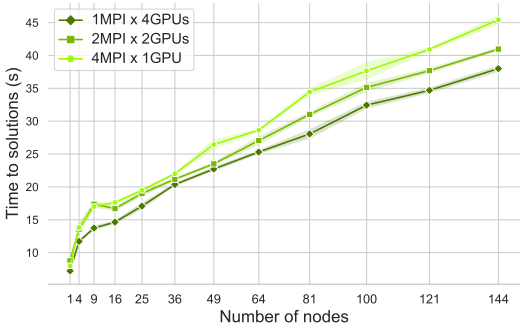
MPI and GPU binding analysis

- Multi-GPU ChASE allows different binding of GPUs to MPI ranks
- We have tested 3 configurations (per node):
  - 1 MPI rank bounded to 4 GPUs (1MPI×4GPUs)
  - 2 MPI ranks bounded to 2 GPUs each (2MPI×2GPUs)
  - 4 MPI ranks each bounded to 1 GPU (4MPI×1GPU)
- Use-case
  - Number of compute nodes is p^2, $p = 1, \ldots, 12$
  - Matrices of type UNIFORM
  - Matrix sizes are $3 \times 10^4 p$ (fixed computational load per node)
  - $nev = 2250$, $nex = 750$
  - 20 repetitions

MPI and GPU binding analysis II



`Filter`'s performance in TFLOPS/node.



Comparison of time-to-solution of ChASE.

Eigen-type tests

Analyze the numerical robustness of ChASE-GPU implementation

Test suite

- Size of test matrices fixed to $20k \times 20k$
- Number of eigenvalues is $nev = 1500$ and additional vectors of search space $nex = 500$
- 20 runs

Testing environment

- Single JURECA DC node
- ChASE-CPU is fixed to 16 MPI rank per node and 8 OpenMP threads per rank
- ChASE-GPU is fixed to 1MPI$\times$4GPUs configuration and 32 OpenMP threads

Eigen-type tests - timing analysis

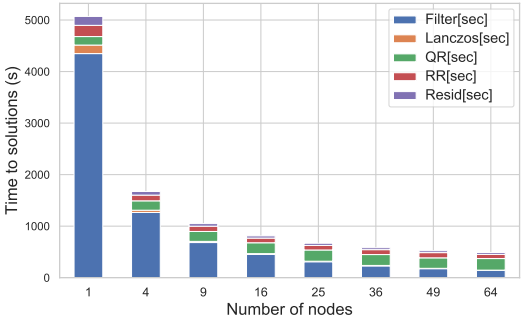
Up: ChASE-CPU, Down: ChASE-GPU.

Matrix	Iter.	Matvecs	Runtime (seconds)					
			`All`	`Lanczos`	`Filter`	`QR`	`RR`	`Resid`
1-2-1	13	466614	272.28 ± 5.28	4.64 ± 0.19	176.46 ± 4.60	31.69 ± 1.27	37.45 ± 1.64	20.99 ± 0.67
Geo	8	285192	165.39 ± 1.86	4.76 ± 0.28	108.02 ± 1.75	19.19 ± 0.59	20.64 ± 1.22	12.14 ± 0.54
Uni	5	163562	101.27 ± 1.98	4.76 ± 0.24	62.17 ± 1.47	12.05 ± 0.53	13.91 ± 0.98	7.97 ± 0.60
Wilk	9	248946	155.44 ± 2.64	4.86 ± 0.96	95.68 ± 1.77	21.53 ± 0.88	20.62 ± 1.25	12.09 ± 0.47

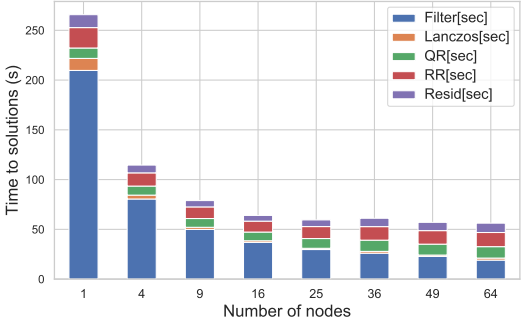
Matrix	Iter.	Matvecs	Runtime (seconds)					
			`All`	`Lanczos`	`Filter`	`QR`	`RR`	`Resid`
1-2-1	13	466614	31.39 ± 0.09	0.58 ± 0.01	14.38 ± 0.02	2.59 ± 0.01	8.41 ± 0.09	5.24 ± 0.04
Geo	8	285192	18.57 ± 0.05	0.58 ± 0.01	8.76 ± 0.02	1.58 ± 0.01	4.58 ± 0.04	2.96 ± 0.02
Uni	5	163562	11.79 ± 0.03	0.58 ± 0.01	5.06 ± 0.00	1.00 ± 0.00	3.11 ± 0.04	1.96 ± 0.02
Wilk	8	246924	17.22 ± 0.05	0.57 ± 0.00	7.63 ± 0.02	1.59 ± 0.00	4.45 ± 0.04	2.90 ± 0.02

Speedup: `All` $8.9\times$, `Filter` $12.7\times$

Strong scaling



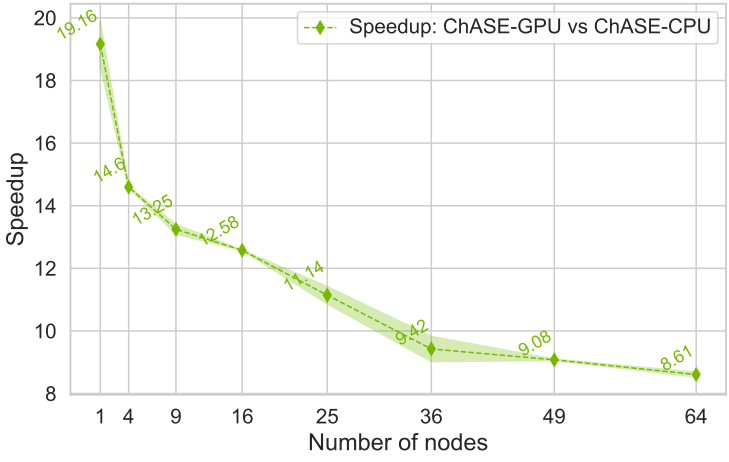
ChASE-CPU



ChASE-GPU

Uniform matrix, $n = 130000$, nev$= 1000$, and nex$= 300$. Data are obtained as the averages of 15 repetitions.

Strong scaling - speedup



Speedup of ChASE-GPU over ChASE-CPU. 15 repetitions

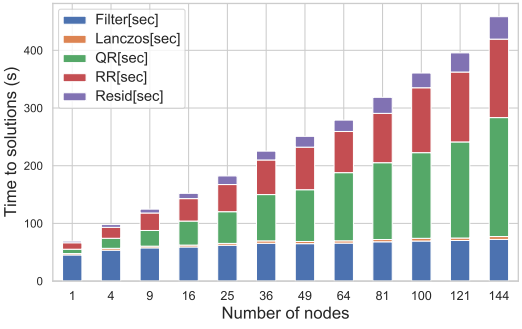
Weak scaling

Simulate the performance on the increasing problem size

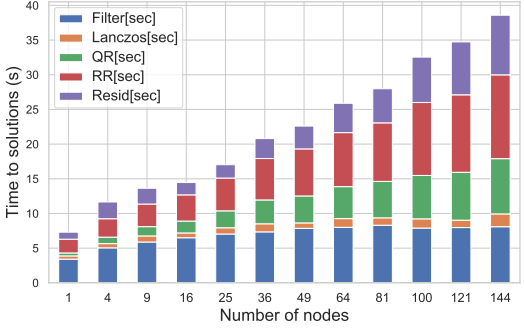
Test configuration

- Test matrices of type `Uniform`
- $n = 30k, 60k, 90k, \ldots, 360k$
- *nev* and *nex* fixed to 2250 and 750
- Number of computes nodes as square numbers $1, 4, 9, \ldots, 144$
- The load per MPI process is constant ($\approx 30k$)

Weak scaling - time analysis



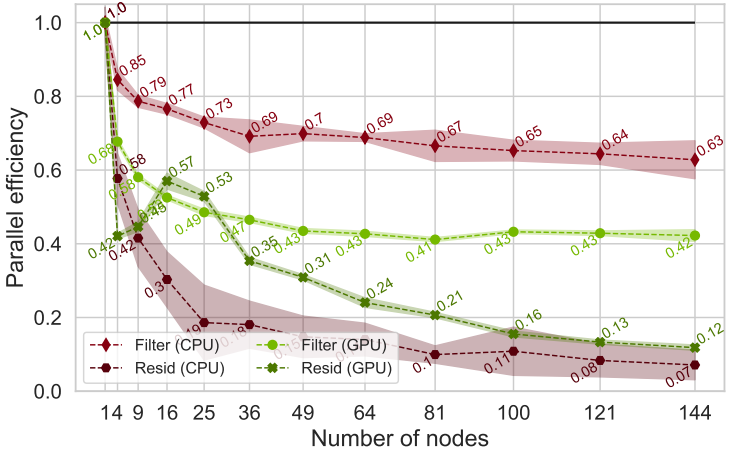
ChASE-CPU



ChASE-GPU

Uniform matrix, n ranging from 30k to 360k, nev= 2250, nex= 750). Data are obtained as the averages of 15 repetitions.

Weak scaling - parallel efficiency



Parallel efficiency of `Filter` and `Resid`. 15 repetitions

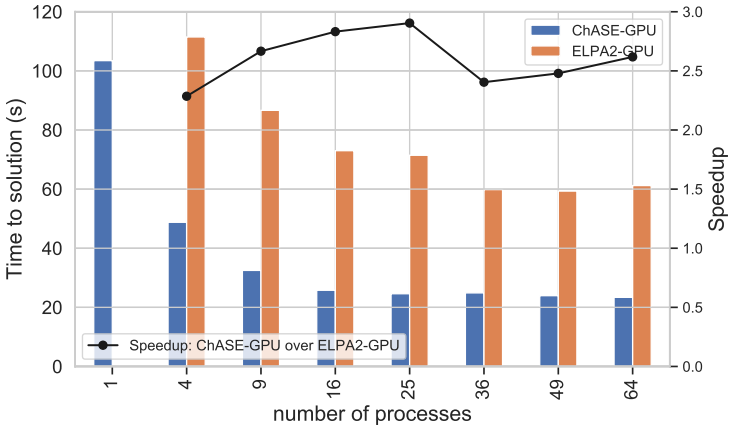
Comparison with other libraries

- Only ELPA2 library provides distributed GPU eigensolvers
- Strong scaling tests on up to 64 nodes comparing ChASE-GPU with ELPA2 with GPU support

ELPA configuration

- Multi-Process Service (MPS) activated
- Number of MPI ranks and GPUs per node is set to 32 and 4, respectively
- Blocks size of block-cyclic distribution is fixed to 16

Execution time



Strong scaling: Time-to-solution and speedup of ChASE-GPU over ELPA2 for solving 76k In_2O_3 Hermitian eigenproblem with nev=800 (nex = 800). Data are obtained as the averages of 15 repetitions.

Outline

Conclusion

- We presented the distributed multi-GPU eigensolver (ChASE) for large-scale symmetric/Hermitian eigenproblems
- The ChASE is extended with a **customized distributed multi-GPU HEMM** used in many parts of the code
- The main performance gain is achieved in `Filter` part showing very good strong and weak scalability
- The new performance bottlenecks are now `QR` and `Rayleigh-Ritz`

Future work

- Increase the cross-platform portability of the code and performance by adding support for novel AMD GPUs.
- Introduce mixed-precision arithmetic to further decrease the computational time.
- Resolve new bottlenecks of the ChASE code such as a per-MPI redundant QR factorization of tall-and-skinny matrices by a customized distributed GPU-based QR solver.
- Continue in optimizing the ChASE code for solving very large scale eigenvalue problems on current PETAscale and future EXAscale supercomputers.

Thank you for your attention

Questions?

HybridScale
UIP-2020-02-4559

ChASE is available on `https://github.com/ChASE-library/ChASE`