ChASE

CHEBYSHEV ACCELERATED SUBSPACE EIGENSOLVER

CHAS(E)ING HERMITIAN DENSE EIGENPROBLEMS WITH SUBSPACE ITERATION

on large scale hybrid platforms with application to DFT

November 16, 2022 | **E. Di Napoli** and X. Wu – e.di.napoli@fz-juelich.de |

OUTLINE

Simulation libraries in Materials Science in the pre-exascale era

Exploiting correlation in plane wave DFT

Algorithm and dependence on HPC kernels

Benchmarking tests

Features and usage

SIMULATION SOFTWARE IN MATERIALS SCIENCE

Some guiding principles

- exploit available **knowledge**.
- increase the **parallelism** of complex tasks.
- facilitate **performance portability**

Facilitating parallelism and HPC

- A **flexible interface** that can accommodate knowledge as input;
- An algorithm design that **avoids inter-node communication**;
- Use of specialized libraries (MKL, cuBLAS, BLIS, etc.) to maximizes the extraction of **many- and multi-core performance**;
- A stable developing team **quickly porting** the core kernels to the latest platforms.

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. 1.1.2 – June 13th 2022
- Reference key: `https://doi.org/10.1145/3313828`
- Reference key: `https://doi.org/10.1145/3539781.3539792`

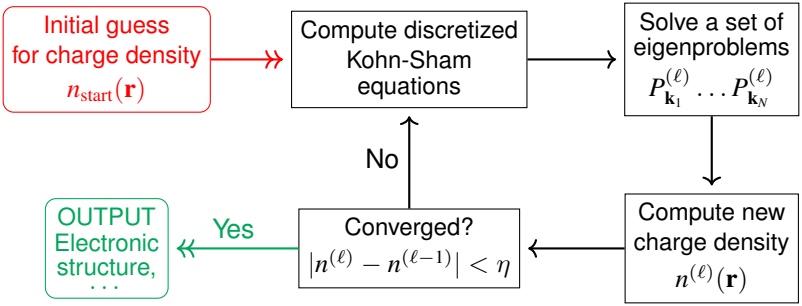
Highlights

- Sequences of dense eigenproblems: exploits correlation between adjacent problems
- Modern C++ interface: depends only on LAPACK and BLAS functions
- Performance portable: excellent strong- and weak-scale performance
- Easy-to-integrate: ready-to-use Fortran to C++ interface

DFT SELF-CONSISTENT FIELD CYCLE

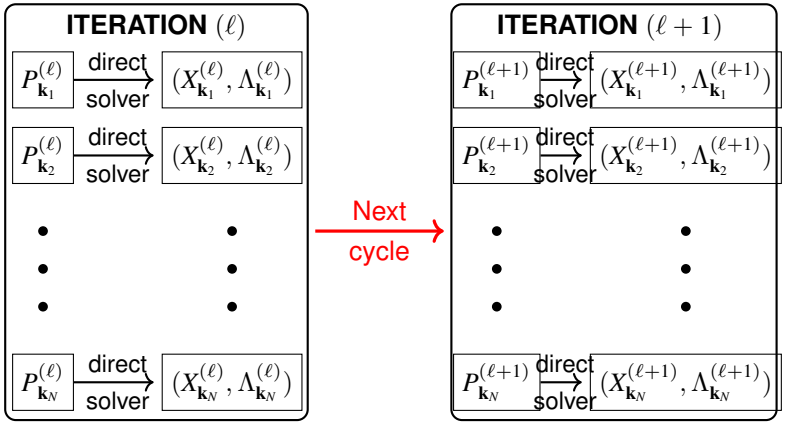
The Schrödinger equation for a all the dgrees of freedom of a multi-atom systems translates into a set of coupled non-linear low-dimensional self-consistent Kohn-Sham (KS) equation

$$\forall\, a \quad \text{solve} \quad \hat{H}_{\mathrm{KS}}\phi_a(\mathbf{r}) = \left(-\frac{\hbar^2}{2m}\nabla^2 + V_0(\mathbf{r})\right)\phi_a(\mathbf{r}) = \epsilon_a\phi_a(\mathbf{r})$$



SEQUENCES OF EIGENPROBLEMS

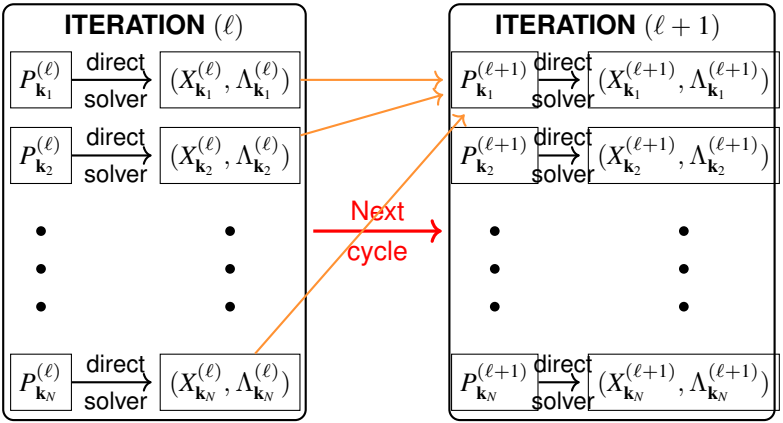
Adjacent iteration cycles



$$X \equiv \{x_1, \ldots, x_n\} \qquad \Lambda \equiv \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$$

SEQUENCES OF EIGENPROBLEMS

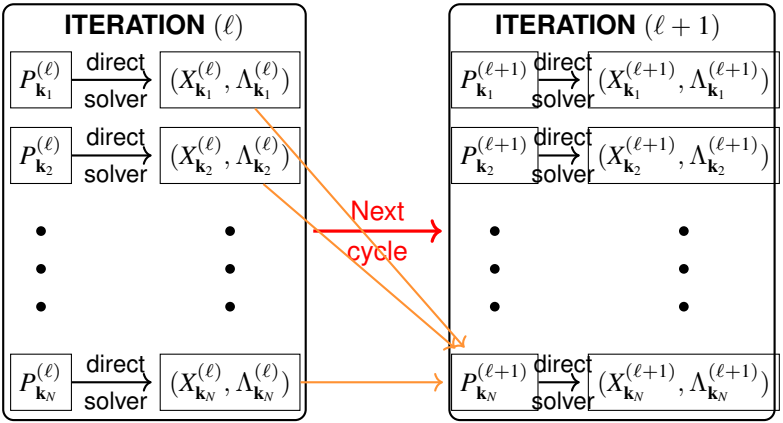
Adjacent iteration cycles



$$X \equiv \{x_1, \ldots, x_n\} \qquad \Lambda \equiv \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$$

SEQUENCES OF EIGENPROBLEMS

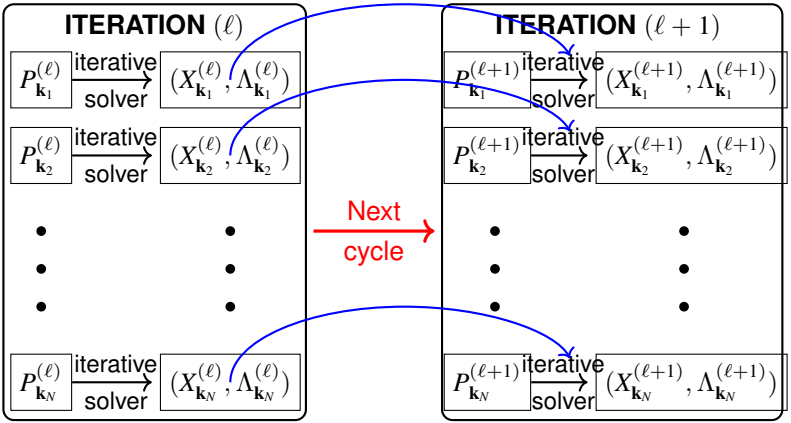
Adjacent iteration cycles



$$X \equiv \{x_1, \ldots, x_n\} \qquad \Lambda \equiv \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$$

AN ALTERNATIVE SOLVING STRATEGY

Adjacent cycles

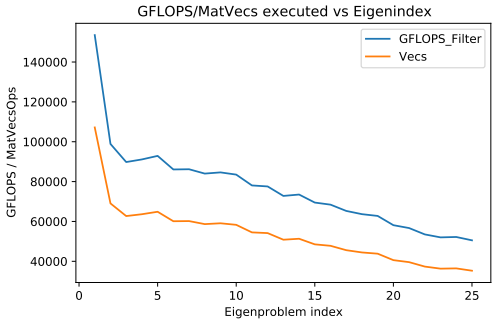


$$X \equiv \{x_1, \ldots, x_n\} \qquad \Lambda \equiv \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$$

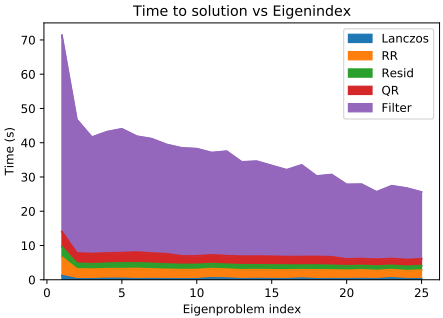
EIGENPROBLEM SEQUENCE

FLAPW matrices: type AuAg DENSE**,** $N = 13379$**, nev** $= 972$ **and nex** $= 40$

Number of operations



Time to solution



Supplementary material

CHEBYSHEV SUBSPACE ITERATION ALGORITHM

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.

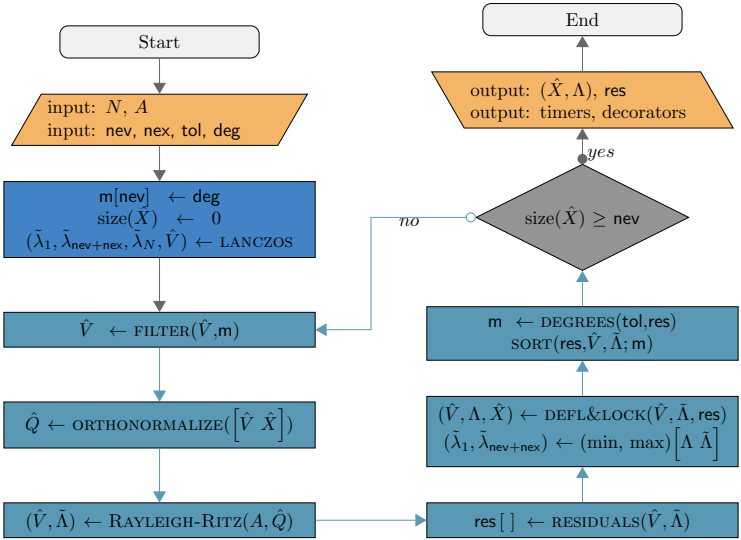
REPEAT UNTIL CONVERGENCE:

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

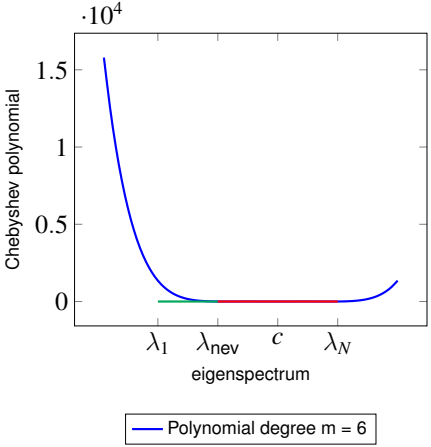
Legend: Original algorithnic contribution, 2D MPI parallel, executed redundantly on each process

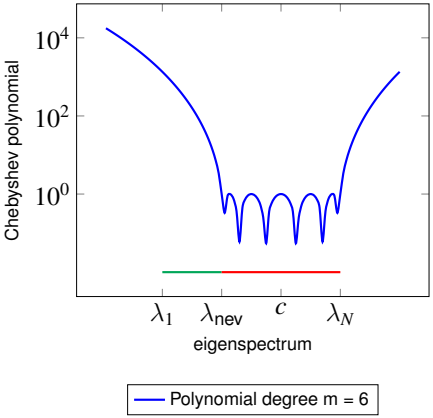
CHEBYSHEV SUBSPACE ITERATION ALGORITHM



DIVIDE AND CONQUER

Chebyshev polynomials





THE CORE OF THE ALGORITHM: CHEBYSHEV FILTER

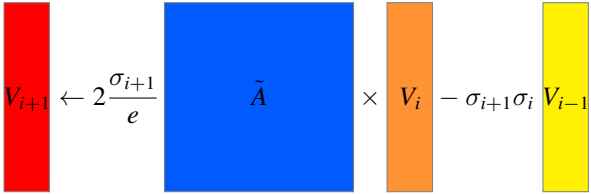
In practice

Three-terms recurrence relation

$$C_{m+1}(t) = 2tC_m(t) - C_{m-1}(t)\,; \qquad m \in \mathbb{N}, \quad C_0(t) = 1, \quad C_1(t) = t$$

$$V_m \doteq p_m(\tilde{A})\, V \quad \text{with} \quad \tilde{A} = A - cI_N$$

FOR: $i = 1 \rightarrow \text{deg} - 1$

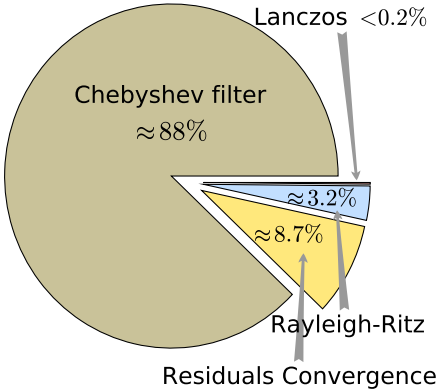


`xHEMM`

END FOR.

WORKLOAD DISTRIBUTION

$Au_{98}Ag_{10}$ - $n = 8{,}970$ - 32 cores.



- `xHEMM` most expensive part
- Parallelizes easily over
  - MPI
  - GPUs
- Good weak scaling
- Recall: Matrix dimensions skewed

PARALLELIZATION OF THE CHEBYSHEV FILTER

Targets

- A simple and efficient scheme for data distribution and communication using MPI
- An economic paradigm that successively performs

$$C \leftarrow \alpha AB + \beta C, \quad B \leftarrow \alpha AC + \beta B. \tag{1}$$

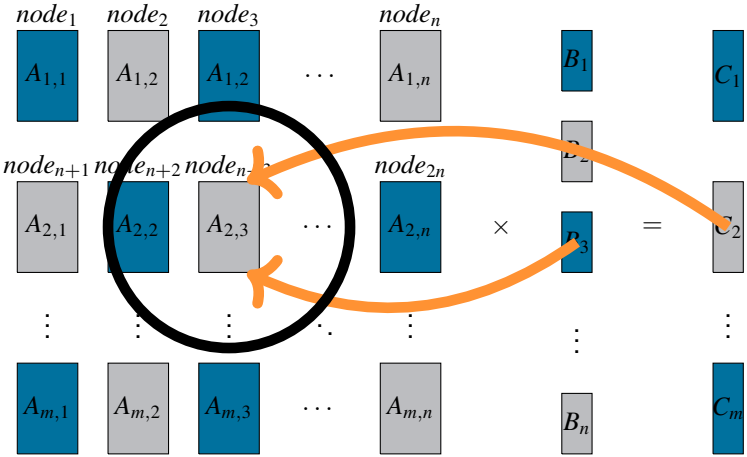
using customized multi-GPU `HEMM` kernel

Desired features

- Develop a scheme for parallelization of the 3-terms recurrence relation Chebyshev filter.
- Harness the power of GPUs.
- Limited GPU memory $\Rightarrow$ multiple GPU nodes
- Minimize communication and redistribution of data.

MATRIX AND VECTORS DISTRIBUTION

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



Distributed HEMM scheme

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 $k = 1, \cdots, n$.

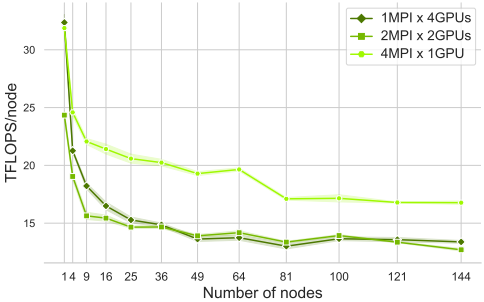
Matrix Name	Spectral Distribution
UNIFORM (UNI)	$\lambda_k = d_{max}(\epsilon + \frac{(k-1)(1-\epsilon)}{n-1})$
GEOMETRIC (GEO)	$\lambda_k = d_{max}\epsilon^{\frac{n-k}{n-1}}$
(1-2-1) (1-2-1)	$\lambda_k = 2 - 2\cos(\frac{\pi k}{n+1})$
WILKINSON (WILK)	All positive, but one, roughly in pairs.

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

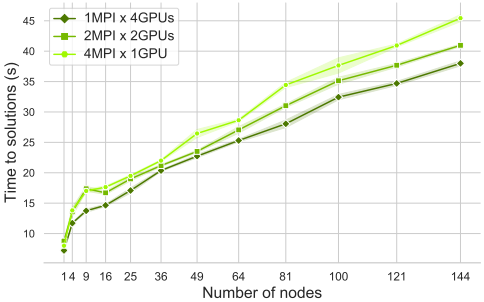
EVALUATING GPU/CPU BINDING CONFIGURATIONS

Uniform matrix: $N = 30000 \times p$, $p = 1, 2, \ldots, 12$, **nev** $= 2250$ **and nex** $= 750$

Filter absolute performance for 1,2 and 4 MPI rank with 32 threads each and 4 GPU



Time to solution for 1,2 and 4 MPI rank with 32 threads each and 4 GPU



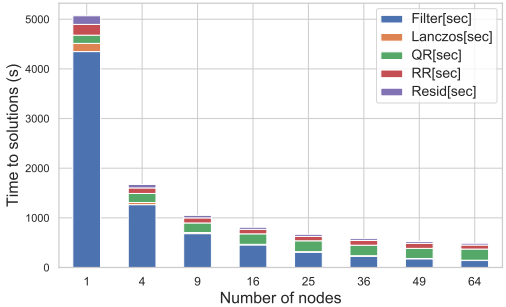
- Only one iteration of subspace;
- FP64 Tensor cores used automatically and selectively (not always) $\Rightarrow$ Absolute performance;
- Filter performs best with 4MPI $\times$ 1 GPU;
- Other BLAS and LAPACK ops performs better with 1MPI $\times$ 4 GPUs.

Sweetspot analysis for CPU $\longrightarrow$ **16 MPI ranks** and **8 threads** per node as optimal.

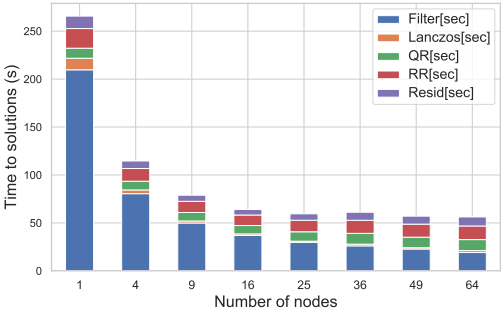
STRONG SCALING

Artificial matrix: type UNIFORM, $N = 130000$, **nev** $= 1000$ **and nex** $= 300$

CPU scaling

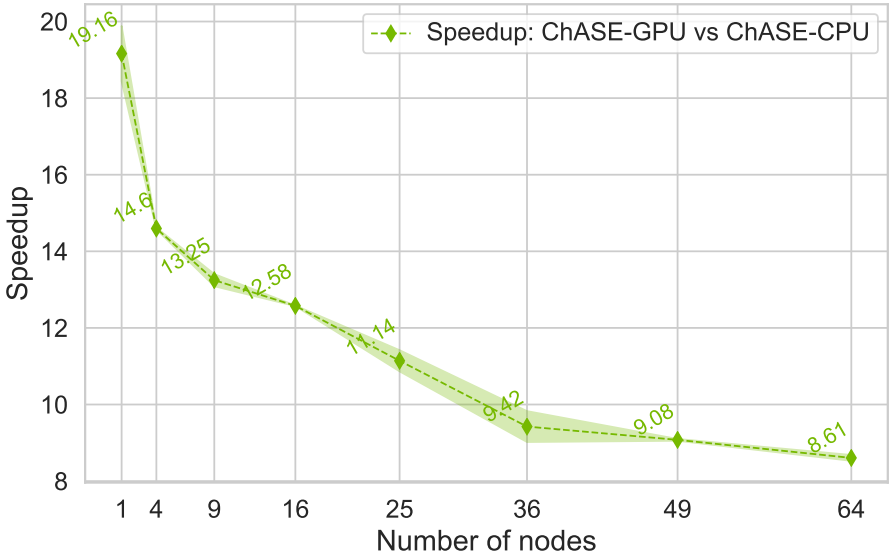


GPU scaling



- 4 $\times$ GPUs with 1 MPI task per node
- Only `HEMM` scales well
- Other operations are done redundantly and become new bottleneck.
- GPU memory is a constraint on size of nev.

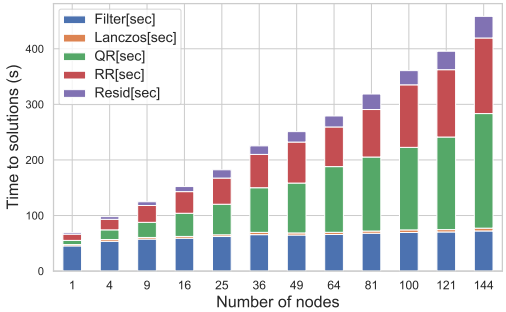
GPU VS CPU


Speedup: ChASE-GPU vs ChASE-CPU
Speedup
Number of nodes
20
18
16
14
12
10
8
1
4
9
16
25
36
49
64
19.16
14.6
13.25
12.58
11.14
9.42
9.08
8.61

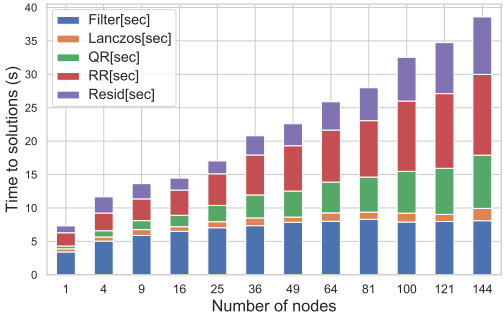
WEAK SCALING

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

CPU scaling



GPU scaling

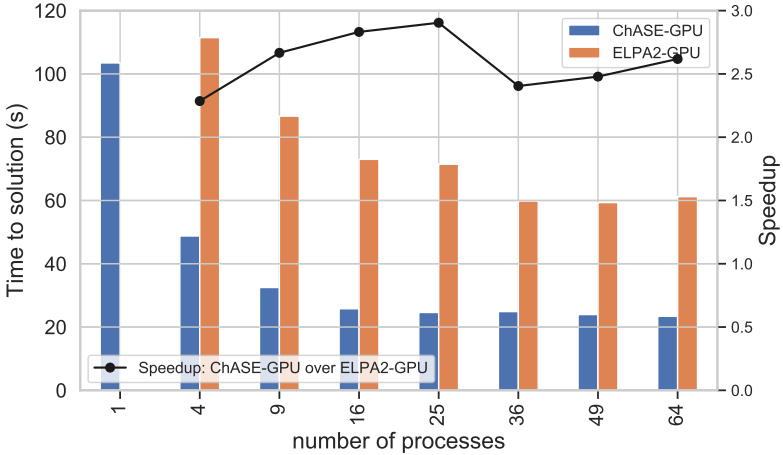


- 4 $\times$ GPUs with 1 MPI task per node;
- ChASE scales linearly;
- Time doubles every time matrix size quadruples (CPU) and triples (GPU);
- Filters scales very well;
- Confirm `QR`, `RR`, `Resid` need a revised parallel computational scheme.

CHASE VS ELPA2

Strong scaling. Hermitian matrix 76k (IN_2O_3). nev=800. Average over 15 repetitions.

- ELPA2 version 2020.11.001
- Compiled with GCC 10.3.0, OpenMPI 4.1.1, Intel MKL 2021.2.0 and CUDA 11.3 with CUDA sm_80
- The MPI core and GPU numbers per node is respectively 32 and 4. Block size is 16.



NEW PARALLEL ALGORITHM

for QR, Rayleigh-Ritz and Residuals

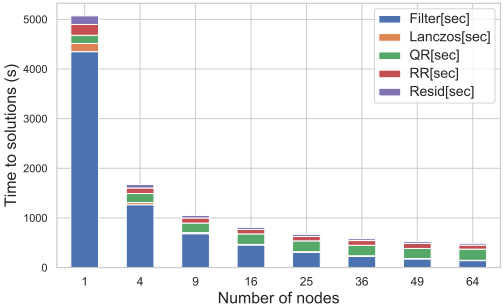
Chase Algorithm

- 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
- Hybrid usage of Householder- and Cholesky-QR for the QR factorization
- Hiding communication with computation within for RR projection and Residual computation
- CPU version only on `devel branch` (GPU implementation under development), soon to be released
- **Much better strong and weak scaling**

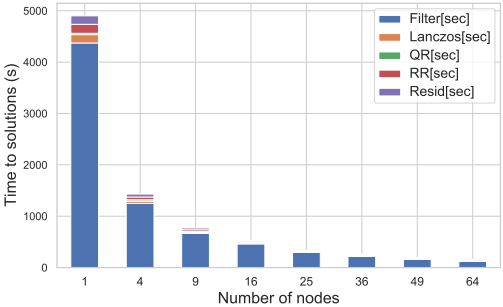
STRONG SCALING

Artificial matrix: type UNIFORM, $N = 130000$, **nev** $= 1000$ **and nex** $= 300$

Old CPU scaling



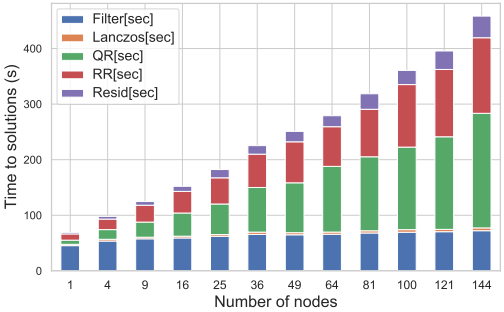
New CPU scaling



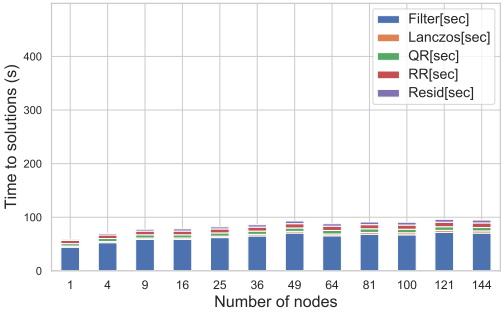
WEAK SCALING

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

Old CPU scaling



New CPU scaling



CHASE ABSTRACT CLASS

Listing 1 Class interface that abstracts the ChASE algorithm from the numerical kernels

```
using T = std::complex<double>;
class Chase {
 public:
  virtual void Start() = 0;                                          // Alg. 1 line 1
  virtual void End() = 0;                                            // Alg. 1 line 16
  virtual void Resd(double *ritzv, double *resd,                     // Alg. 1 line 8
                    size_t fixednev) = 0;
  virtual void Lock(size_t new_converged) = 0;                       // Alg. 3 line 7
  virtual void QR(size_t fixednev) = 0;                              // Alg. 1 line 5
  virtual void RR(double *ritzv, size_t block) = 0;                  // Alg. 2
  virtual void HEMM(size_t nev, T alpha, T beta, size_t s) = 0;      // Alg. 4 line 9
  virtual void Lanczos(size_t k, double *upperb) = 0;                // Alg. 6 line 3
  virtual void Lanczos(size_t M, size_t j, double *upperb,           // Alg. 6 line 8
                       double *ritzv, double *Tau,
                       double *ritzV) = 0;
  virtual void LanczosDos(size_t idx, size_t m, T *ritzVc) = 0;      // Alg. 6 line 13
  virtual void Shift(T c, bool isunshift = false) = 0;               // A − cI_n
  virtual void Swap(size_t i, size_t j) = 0;                         // Swap V̂_{:,i} and V̂_{:,j}

  /* ommited Getters */
};
```

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.

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)

ON LINE DOCUMENTATION

https://chase-library.github.io/ChASE/index.html

ChASE
CHEBYSHEV ACCELERATED SUBSPACE EIGENSOLVER

v1.0.0

Search docs

Docs » ChASE's Documentation View page source

ChASE's Documentation

INTRODUCTION

USER DOCUMENTATION

LESSONS LEARNED

...in the last 5 years

1. Exploiting domain knowledge may accelerate computation.
2. flexible abstract interface separating algorithm from implementation $\longrightarrow$ facilitates porting performance;
3. Extracting node-level performance using specialized kernels is not trivial $\longrightarrow$ sweetspot analysis or detailed heuristics are necessary to understand the optimal configuration;
4. 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);

OUTLOOK

- Porting of new QR/RR/Resid parallelization to multi-GPU $\longrightarrow$ January 2023
- Porting to AMD GPUs with aim at new Jupyter exascale modular cluster $\longrightarrow$ June 2023
- Nice to have: integration with ELSI platform
- Porting to ARM-based platforms $\longrightarrow$ sometimes in 2023
- Extension to interior eigenproblems through rational spectral filters $\longrightarrow$ September 2023
- Long term: extension to non-Hermitian eigenproblems.

References

EDN, Blügel, Bientinesi – `http://dx.doi.org/10.1016/j.cpc.2012.03.006` (2012)

Berljafa, Wortmann, EDN – `https://10.1002/cpe.3394` (2014)

Winkelmann, Springer, EDN – `https://doi.org/10.1145/3313828` (2019)

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

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

SEQUENCES OF EIGENPROBLEMS

Definitions and solving strategies

Definition: Eigenproblem sequence

A sequence of eigenproblems is a finite and index-ordered set of problems $\{P\}_N \doteq P^{(1)} \dots P^{(\ell)} \dots P^{(N)}$ with same size $= n$ such that the eigenpairs of $P^{(\ell)}$ are used (directly or indirectly) to initialize $P^{(\ell+1)}$.

Current solving strategy

- The set of generalized eigenproblems $P^{(1)} \dots P^{(\ell)} P^{(\ell+1)} \dots P^{(N)}$ is handled as a set of disjoint problems $N \times P$;
- Each problem $P^{(\ell)}$ is solved independently using a direct solver as a black-box from a standard library (i.e. ScaLAPACK).

CORRELATION BETWEEN EIGENPROBLEMS

Definition and solving strategies

Definition: Correlation

Two adjacent problems $P^{(\ell+1)}$ and $P^{(\ell)}$ are said to be correlated when the eigenpairs $(X^{(\ell+1)}, \Lambda^{(\ell+1)})$ have some relation with the eigenpairs $(X^{(\ell)}, \Lambda^{(\ell)})$.

CORRELATION BETWEEN EIGENPROBLEMS

Definition and solving strategies

Definition: Correlation

Two adjacent problems $P^{(\ell+1)}$ and $P^{(\ell)}$ are said to be correlated when the eigenpairs $(X^{(\ell+1)}, \Lambda^{(\ell+1)})$ have some relation with the eigenpairs $(X^{(\ell)}, \Lambda^{(\ell)})$.

Uncovering the correlation $\longrightarrow$ extracting information from simulations

- Extracted the matrices of eigenproblems $P^{(1)}, \ldots, P^{(N)}$ from the FLAPW code by running a full simulation;
- Computed the solutions of the full sequence,
  - collected data on **angles** b/w eigenvectors of adjacent eigenproblems;

$$\Theta^{(\ell)}_{\mathbf{k}_i} \equiv \{\theta_1, \ldots, \theta_n\} = \mathrm{diag}\left(\mathbb{1} - \langle X^{(\ell-1)}_{\mathbf{k}_i}, \tilde{X}^{(\ell)}_{\mathbf{k}_i}\rangle\right)$$

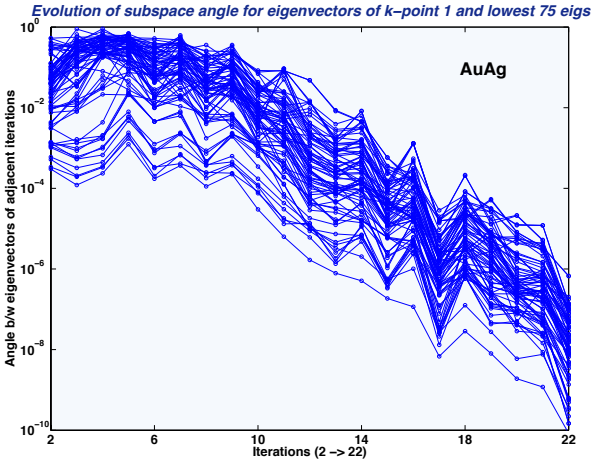
  - uncovered **evolution** of eigenvectors along the sequence

$$\text{for fixed } \mathbf{k}_i \quad \theta^{(2)}_j \gtrsim \theta^{(3)}_j \gtrsim \cdots \gtrsim \theta^{(N)}_j : \quad \theta^{(2)}_j \gg \theta^{(N)}_j$$

ANGLES EVOLUTION

An example

Example: a metallic compound at fixed $\mathbf{k}$



THE CORE OF THE ALGORITHM: CHEBYSHEV FILTER

The basic principle

Theorem

Let $|\gamma| > 1$ and $\mathbb{P}_m$ denote the set of polynomials of degree smaller or equal to m. Then the extremum

$$\min_{p \in \mathbb{P}_m, p(\gamma)=1} \max_{t \in [-1,1]} |p(t)|$$

is reached by

$$p_m(t) \doteq \frac{C_m(t)}{C_m(\gamma)}.$$

where C_m is the Chebyshev polynomial of the first kind of order m, defined as

$$C_m(t) = \begin{cases} \cos\left(m \arccos(t)\right), & t \in [-1, 1]; \\ \cosh\left(m \operatorname{arccosh}(t)\right), & |t| > 1. \end{cases}$$

SUBSPACE ITERATION

Power Iteration: Given a generic vector $v = \sum_{i=1}^{n} s_i x_i$

$$v^m = A^m v = \sum_{i=1}^{n} s_i A^m x_i = \sum_{i=1}^{n} s_i \lambda_i^m x_i = s_1 x_1 + \sum_{i=2}^{n} s_i \left(\frac{\lambda_i}{\lambda_1}\right)^m x_i \sim \boxed{s_1 x_1}$$

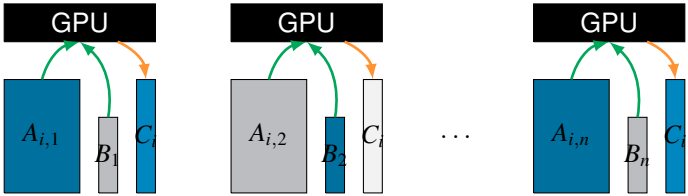
Subspace iteration + Chebyshev polynomials:

$$\begin{aligned}
v^m = p_m(A)v &= \sum_{i=1}^{n} s_i\, p_m(A) x_i = \sum_{i=1}^{n} s_i\, p_m(\lambda_i) x_i \\
&\approx \sum_{i=1}^{\mathsf{nev}} s_i \frac{C_m\left(\frac{\lambda_i - c}{e}\right)}{C_m\left(\frac{\gamma - c}{e}\right)} x_i + \sum_{j=\mathsf{nev}+1}^{n} s_j x_j \\
\text{Reorthogonalization} &+ \text{Rayleigh} - \text{Ritz} \\
&\approx \sum_{i=1}^{\mathsf{nev}} \left(s_i x_i + \sum_{j=\mathsf{nev}+1}^{n} S_j^i \frac{1}{|\rho_i|^m} x_j \right) \\
&\sim \boxed{\textstyle\sum_{i=1}^{\mathsf{nev}} s_i x_i}
\end{aligned}$$

MPI SCHEME

Step 1

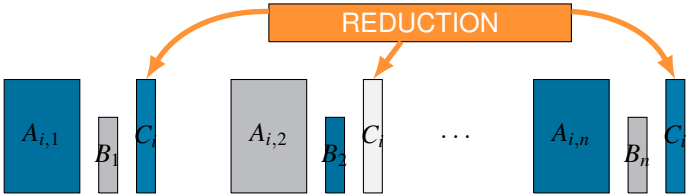
Calculate AB on the GPU, return it to CPU and save in temporary C_{tmp}.



MPI SCHEME

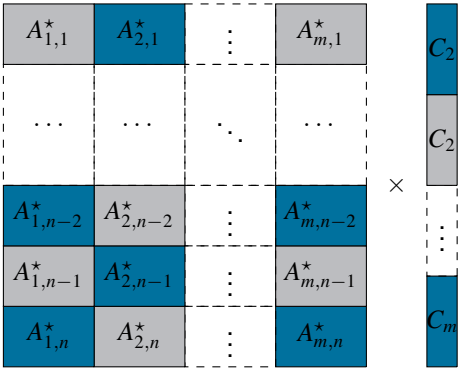
Step 2

Perform reduction (summation) on nodes in each row. Then save $\alpha C_{tmp} + \beta C$ in C.



NEXT STEP:

Redistribution of C is avoided thanks to the simple observation that $A = A^H$

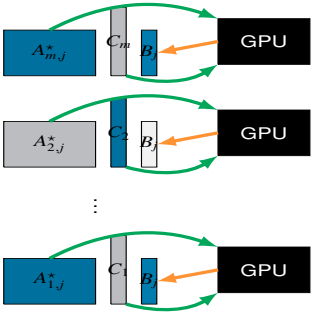


Repeat the previous steps for $\alpha A^H C + \beta B$

MPI SCHEME

Step 3

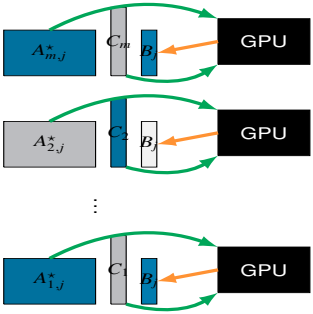
Calculate AC on the GPU, return it to CPU and save in temporary B_{tmp}.



MPI SCHEME

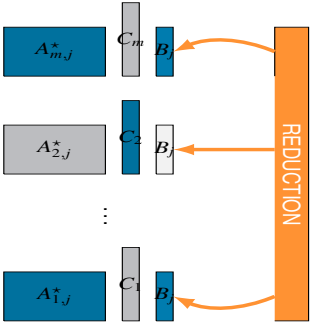
Step 3

Calculate AC on the GPU, return it to CPU and save in temporary B_{tmp}.



Step 4

Perform reduction on nodes in each column. Then save $\alpha B_{tmp} + \beta B$ in B.



MPI SCHEME: RECAP

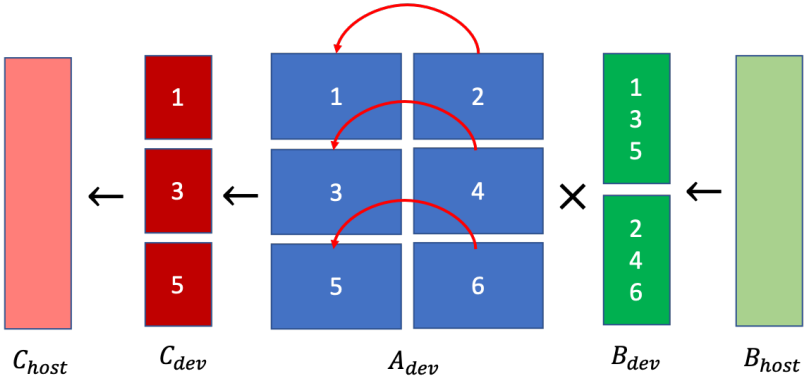
- Steps 1-4 describe two cycles of Chebyshev iteration.
- Performing 3-terms recurrence relation within the Chebyshev iterations relies on alternating between both kinds of cycles.
- Cycle 1: Perform $A \times B$, and then reduce across every row of the processing grid.
- Cycle 2: Perform $A^\star \times C$, and then reduce on every column of the processing grid.
- Most of the communication is spent in a `MPI_Allreduce`.

Back to presentation

MULTI-GPU MATRIX MULTIPLICATION SCHEME

Guiding principle

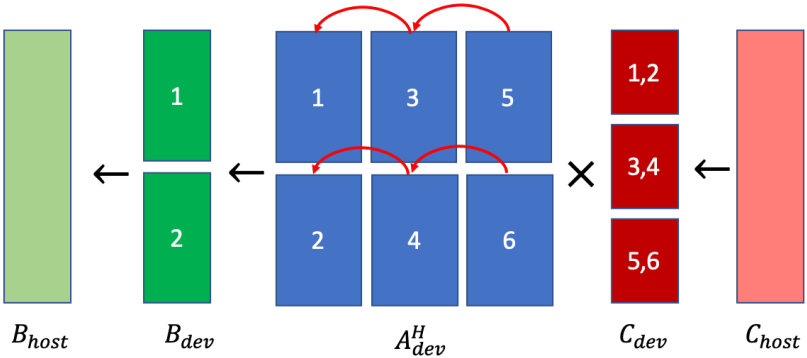
- The distribution of $A_{i,j}$ on GPUs plays a guiding role
- The distribution of B_j and C_i is a result of the distribution of $A_{i,j}$.



MULTI-GPU MATRIX MULTIPLICATION SCHEME

Guiding principle

- The distribution of $A_{i,j}$ on GPUs plays a guiding role
- The distribution of B_j and C_i is a result of the distribution of $A_{i,j}$.



EXTRACTING PERFORMANCE AT THE NODE-LEVEL

Sweetspot analysis for weak scaling benchmark tests on JURECA-DC

nodes	tasks	threads	ChASE time (s)	Lanczos (s)	Filter (s)	QR (s)	RR (s)	Resid (s)
1	1	128	351.81	27.43	275.78	8.30	24.53	13.61
1	2	64	161.53	13.48	126.92	3.93	9.54	6.57
1	4	32	93.41	6.75	69.87	3.85	7.70	4.68
1	8	16	75.93	3.81	53.54	5.30	8.06	4.95
1	16	8	83.87	5.95	53.45	8.50	12.78	3.05
1	32	4	102.05	2.46	55.86	16.57	23.49	3.59
1	64	2	146.32	2.97	59.20	32.64	45.36	6.10
1	128	1	272.09	3.96	65.46	101.95	93.12	7.57
16	1	128	991.63	37.63	305.02	25.72	426.70	194.37
16	2	64	219.98	20.10	136.22	16.55	20.60	25.40
16	4	32	127.43	10.84	72.88	18.02	18.61	6.53
16	8	16	121.25	8.90	56.49	29.47	20.68	5.42
16	16	8	160.03	7.54	57.28	46.86	40.87	7.35
16	32	4	239.42	2.97	56.03	96.70	73.71	9.93
16	64	2	412.08	3.47	57.51	191.06	148.86	11.10
16	128	1	933.09	4.90	59.57	543.47	304.30	20.81

1 node $N = 30{,}000.0$, 16 nodes $N = 120{,}000.0$, nev $= 2250$ and nex $= 750$.

COMPARING CHASE-CPU AND CHASE-GPU

with artificial matrices

Matrix size: $20\text{k} \times 20\text{k}$. nev and nex are 1500 and 500, respectively. Statistics for each test are obtained over 20 runs. **Instability of** `cusolverXgeqrf`.

Matrix	Iter.	Matvecs	**ChASE-CPU** Runtime (seconds)				
			All	Filter	QR	RR	Resid
1-2-1	13	466614	272.28 ± 5.28	176.46 ± 4.60	31.69 ± 1.27	37.45 ± 1.64	20.99 ± 0.67
GEO	8	285192	165.39 ± 1.86	108.02 ± 1.75	19.19 ± 0.59	20.64 ± 1.22	12.14 ± 0.54
UNI	5	163562	101.27 ± 1.98	62.17 ± 1.47	12.05 ± 0.53	13.91 ± 0.98	7.97 ± 0.60
WILK	9	248946	155.44 ± 2.64	95.68 ± 1.77	21.53 ± 0.88	20.62 ± 1.25	12.09 ± 0.47

Matrix	Iter.	Matvecs	**ChASE-GPU** Runtime (seconds)				
			All	Filter	QR	RR	Resid
1-2-1	13	466614	31.39 ± 0.09	14.38 ± 0.02	2.59 ± 0.01	8.41 ± 0.09	5.24 ± 0.04
GEO	8	285192	18.57 ± 0.05	8.76 ± 0.02	1.58 ± 0.01	4.58 ± 0.04	2.96 ± 0.02
UNI	5	163562	11.79 ± 0.03	5.06 ± 0.00	1.00 ± 0.00	3.11 ± 0.04	1.96 ± 0.02
WILK	8	246924	17.22 ± 0.05	7.63 ± 0.02	1.59 ± 0.00	4.45 ± 0.04	2.90 ± 0.02

PARALLEL EFFICIENCY