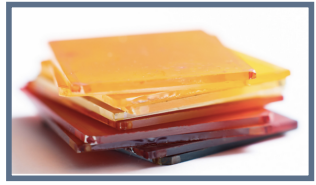
THE CHASE LIBRARY FOR LARGE HERMITIAN EIGENVALUE PROBLEMS

April 17, 2018 | **E. Di Napoli**, J. Winkelmann, A. Schleife |

Theoretical Spectroscopy: Oxides, Perovskites

Optoelectronics and semiconductor technology:

- Lasers and light-emitting diodes
- Transparent electronics
- Tunable optical properties



$CH_3NH_3PbI_3$

Energy-related applications:

- Photocatalytic water splitting
- Photovoltaic absorbers
- Transparent electrodes for PV

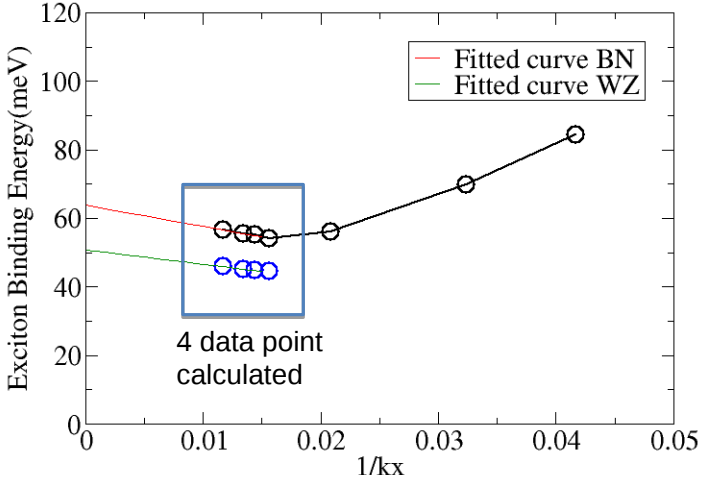
Excitonic effects: solution of the Bethe-Salpeter equation

- Leads to eigenvalue problem (excitonic Hamiltonian)
- Huge matrix: Rank typically > 50,000
- Time-propagation approach to calculate the dielectric function

- Excellent description of the optical properties of the oxides

→ predictive power (e.g. for perovskites)

Exciton binding energies: BN ZnO



- Standard approach: Linear extrapolation after “turnaround point”
- Larger energy cutoff desirable (but unaffordable)
- More complicated oxides desirable, e.g. for materials design

Exciton binding energies: BN ZnO

- Layered meta-stable BN structure of ZnO reported for ultra-thin films
- Computation of (converged) binding energies computationally expensive
- Particularly difficult: Convergence with respect to ***k***-point sampling

# k-points	Rank of matrix	Total # of matrix elements	Time consumed	Memory required	Nodes used
10945	82499	6.8×10^{9}	1.5 hours	50.7 GB	8
12713	96399	9.3×10^{9}	2 hours	69.2 GB	8
16299	124281	1.5×10^{10}	2 hours	115.1 GB	16
25367	195281	3.8×10^{10}	5.5 hours	284.1 GB	16

- Cost increases enormously as k-points number increases
- Convergence barely achieved, despite very simple system (4 atoms per unit cell)

COMPUTATIONAL PROBLEM

Extremal Hermitian Eigenvalue Problem

- Needed: $\mathcal{O}(100)$ lowest eigenvalues (exciton binding energies) of
- $H^\dagger = H \in \mathbb{C}^{n\times n}$, dense
- $20,000 \leq n \leq 200,000$ (up to $1,000,000$)
- Single Precision Complex

- Current eigensolver is based on Kalkreuther-Simma Conjugate-Gradient (KSCG) algorithm;
- it is based on single `mat-vec` operations (i.e. `xGEMV`);
- the computationally intense parts are memory bound.

Desiderata

- Needed: increase parallel efficiency, scalability and performance;
- Weak scaling of particular interest;
- Desired: exploit many-core platforms (e.g. GPUs on Blue Waters);
- Compute bound calculations.

THE CHASE ALGORITHM

Subspace Iteration with Rayleigh-Ritz projection

- Choose an initial system of vectors $X^0 = [x_1, \ldots, x_m]$.
- Perform successive multiplication $X^k := AX^{k-1}$.
- Every once in a while orthonormalize column-vectors in X^k.
- Compute Rayleigh quotient
- Solve reduced problem

ChASE Eigensolver

- Substitute $A^k X \longrightarrow p(A)X$.
- Chebyshev filter improves the rate of convergence.

CHASE PSEUDOCODE

INPUT: Hamiltonian H, `tol`, `deg` — OPTIONAL: approximate eigenvectors Z_0, extreme eigenvalues $\{\lambda_1, \lambda_{\text{NEV}}, \lambda_{\text{MAX}}\}$.

OUTPUT: NEV wanted eigenpairs (Λ, W).

1. Lanczos DoS step. Identify the bounds for the **eigenspectrum interval** corresponding to the wanted eigenspace.

REPEAT UNTIL CONVERGENCE:

2. Chebyshev filter. **Filter** a block of vectors $W \longleftarrow Z_0$.
3. Re-orthogonalize the vectors outputted by the filter; $W = QR$.
4. Compute the **Rayleigh quotient** $G = Q^{\dagger}HQ$.
5. Compute the primitive Ritz pairs (Λ, Y) by solving for $GY = Y\Lambda$.
6. Compute the approximate Ritz pairs $(\Lambda, W \leftarrow QY)$.
7. Check which one among the Ritz vectors converged.
8. Deflate and lock the converged vectors.

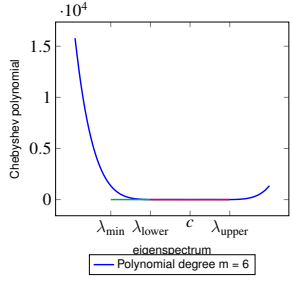
END REPEAT

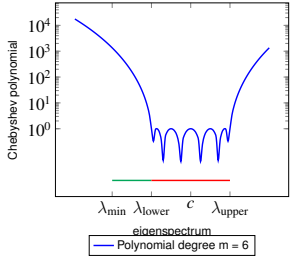
THE CORE OF THE ALGORITHM: CHEBYSHEV FILTER

Chebyshev polynomials

A generic vector $v = \sum_{i=1}^{n} s_i x_i$ is very quickly aligned in the direction of the eigenvector corresponding to the extremal eigenvalue λ_1

$$
\begin{aligned}
v^m = p_m(H)v &= \sum_{i=1}^{n} s_i\, p_m(H) x_i = \sum_{i=1}^{n} s_i\, p_m(\lambda_i) x_i \\
&= s_1 x_1 + \sum_{i=2}^{n} s_i \frac{C_m\left(\frac{\lambda_i - c}{e}\right)}{C_m\left(\frac{\lambda_1 - c}{e}\right)} x_i \;\sim\; \boxed{s_1 x_1}
\end{aligned}
$$





THE CORE OF THE ALGORITHM: CHEBYSHEV FILTER

In practice

Three-terms recurrence relation

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

$$Z_m \doteq p_m(\tilde{H})\, Z_0 \quad \text{with} \quad \tilde{H} = H - cI_n$$

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

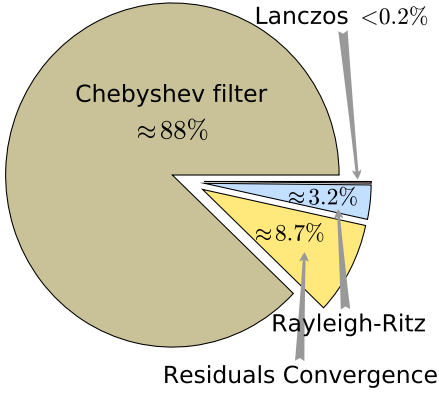
$$Z_{i+1} \leftarrow 2\frac{\sigma_{i+1}}{e} \tilde{H} \times Z_i - \sigma_{i+1}\sigma_i Z_{i-1}$$

xGEMM

END FOR.

WORKLOAD DISTRIBUTION

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



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

ESTIMATING THE INTERVAL BOUNDS

Whenever λ_1, λ_{lower}, and λ_{upper} are unknown, ChASE uses a Density of States (DoS) based of repeated Lanczos processes with few steps [Lin et al. SIREV 58, 34 2016].

1. Compute the DoS

$$\phi_\sigma(t) = \frac{1}{n_{\text{vec}}} \sum_{\ell=1}^{\text{vec}} \left[\frac{1}{n} \sum_{k=0}^{M} \left(\tau_k^{(\ell)} \right)^2 g_\sigma(t - \theta_k^{(\ell)}) \right]$$

2. Compute the cumulative DoS and estimate λ_{lower} such that $\frac{\text{nev}+\text{nex}}{n} = \int_{-\infty}^{\lambda_{\text{lower}}} \phi_\sigma(t)\, dt$.
3. Bounds on λ_1 and λ_{upper} from one Lanczos process.

Typical values used in ChASE

- Width of the Gaussian $\sigma = 0.25$
- Lanczos steps $M = 20 \div 25$
- Number of random vectors $n_{\text{vec}} = 3 \div 5$

POLYNOMIAL DEGREE OPTIMIZATION

Definition

The **convergence ratio** for the eigenvector x_i corresponding to eigenvalue $\lambda_i \notin [\alpha, \beta]$ is defined as

$$\tau(\lambda_i) = |\rho_i|^{-1} = \min_{\pm} \left| \frac{\lambda_i - c}{e} \pm \sqrt{\left(\frac{\lambda_i - c}{e}\right)^2 - 1} \right|.$$

The further away λ_i is from the interval $[\alpha, \beta]$ the smaller is $|\rho_i|^{-1}$ and the faster the convergence to x_i is.

For a set of input vectors $V = \{v_1, v_2, \ldots, v_{\text{nev}}\}$

Residuals are a function of m and $|\rho|$

$$\text{Res}(v_i^m) \sim \quad \text{Const} \times \left|\frac{1}{\rho_i}\right|^m \qquad 1 \le i \le k.$$

$$\text{Res}(v_i^{m+m_0}) \approx \quad \text{Res}(v_i^{m_0}) \left|\frac{1}{\rho_i}\right|^m \quad \Rightarrow \quad m_i \ge \ln \left| \frac{\text{TOL}}{\text{Res}(v_i^{m_0})} \right| / \ln |\rho_i|$$

CHASE PSEUDOCODE (OPTIMIZED)

2. Chebyshev filter. Initial filter $W \longleftarrow Z_0$. with DEG$= m_0$.
3. Re-orthogonalize $W = QR$ & compute the Rayleigh quotient $G = Q^\dagger HQ$.
4. Solve the reduced problem $GY = Y\Lambda$ and compute the approximate Ritz pairs $(\Lambda, W \leftarrow QY)$ and store their residuals $\mathrm{Res}(w_i)$.

REPEAT UNTIL CONVERGENCE:

1. Optimizer. Compute the polynomial degrees $m_i \geq \ln \left| \frac{\mathrm{TOL}}{\mathrm{Res}(w_i)} \right| / \ln |\rho_i|$.
2. Chebyshev filter. Filter $W \longleftarrow Z_0$ with DEG$= m_i$.
3. Re-orthogonalize $W = QR$ & compute the Rayleigh quotient $G = Q^\dagger HQ$.
4. Solve the reduced problem $GY = Y\Lambda$ and compute the approximate Ritz pairs $(\Lambda, W \leftarrow QY)$.
5. lock the converged vectors.
6. Store the residuals $\mathrm{Res}(w_i)$ of the unconverged vectors.

END REPEAT

ONE PROJECT — TWO FRONTS

Towards a “new” ChASE library

- √ Templating ChASE for SP;
- √ Implementing a distributed CPU/GPU parallelization for ChASE inner kernels;
- → Refine ChASE node-level parallelism with multiple GPUs;
- √ Porting ChASE (CPU and GPU) to Blue Waters

Integrating ChASE with Jena BSE code

- → Reconfigure Jena BSE package to initialize matrices with a more efficient I/O;
- → Integrate ChASE with BSE;
  - √ Accuracy and algorithmic hurdles
  - √ Single vs double precision
  - Parallel I/O
- Tuning ChASE for Blue Waters and the BSE application.

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 */
};
```

CHASE BLAS

Listing 2 ChaseBLAS: an implementation of ChASE's kernels with BLAS+LAPACK

```
using T = complex<double>;
class ChaseBLAS : public chase::Chase {
 public:
  ChaseBLAS(size_t N, size_t nev, size_t nex, T *H, T *V, T *W, double *ritzv,
            double *resid)
      : N_(N), nev_(nev), nex_(nex), locked_(0), H_(H), approxV_(V),
        workspace_(W), ritzv_(ritzv), resid_(resid), config_(N_, nev_, nex_) {}

  void HEMM(size_t block, T alpha, T beta, size_t s) override {
    zhemm('L', 'L', N_, block, &alpha, H_, N_, approxV_ + N_ * (locked_ + s),
          N_, &beta, workspace_ + N_ * (locked_ + s), N_);
    swap(approxV_, workspace_);
  };
  void Lock(size_t new_converged) override {
    memcpy(workspace_ + locked_ * N_, approxV_ + locked_ * N_,
           N_ * (new_converged) * sizeof(T));
    locked_ += new_converged;
  };
  /* Additional functions from Listing 1 ommited for conciseness */
 private:
  int N_, nev_, nex_, locked_;
  T *A_, *approxV_, *workspace_;
  double *ritzv_, resid_;
  ChaseConfig<T> config_;
};
```

CONFIGURATION PARAMETERS

Category	Parameter	`ChaseConfig` member routine	Default value
General see Alg. 1	N	*constructor*	N/A
	nev	*constructor*	N/A
	nex	*constructor*	N/A
	approx	`void SetApprox(bool);`	`false`
	tol	`void SetTol(double);`	`1e-10` (`1e-5`)
	maxIter	`void SetMaxIter(size_t);`	`25`
Chebyshev Filter see Alg. 5 and 4	optim	`void SetOpt(bool);`	`false`
	degExtra	`void SetDegExtra(size_t);`	`2`
	degMax	`void SetMaxDeg(size_t);`	`36` (`18`)
	deg	`void SetDeg(size_t);`	`20` (`10`)
Spectral Estimates see Alg. 6	k	`void SetLanczosIter(size_t);`	`25`(`12`)
	n_{vec}	`void SetNumLanczos(size_t);`	`4`

EXPERIMENTAL SETUP

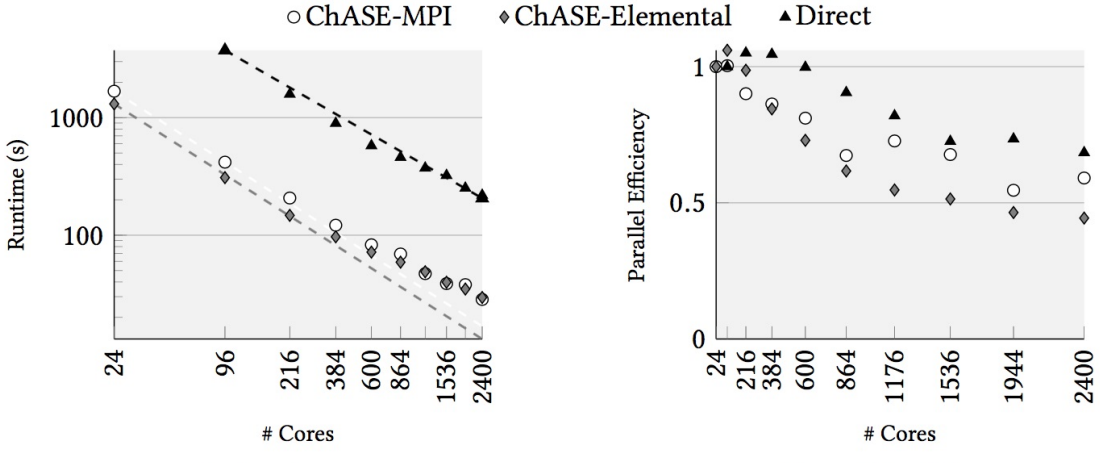
Table: Matrices used in scaling experiments

	# Nodes	# Cores	n	nev	nex	$\frac{n^2}{\#\text{ Cores}}$
NaCl	4	96	3893	256	51	N/A
	25	600	9273	256	51	N/A
AuAg	25	600	13,379	972	194	N/A
BSE	9	216	22,360	100	20	2,314,674
	16	384	32,976	100	20	2,831,814
	36	864	47,349	100	20	2,594,823
	64	1536	62,681	100	20	2,557,882
	100	2400	76,674	100	20	2,449,542

Tests were performed on the JURECA cluster.

- 2 Intel Xeon E5-2680 v3 Haswell – Up to $0.96 \div 1.92$ TFLOPS DP/SP;
- 2 x NVIDIA K80 (four devices) – Up to $2.91 \div 8.74$ TFLOPS DP/SP.

STRONG SCALING



EXPERIMENTAL SETUP

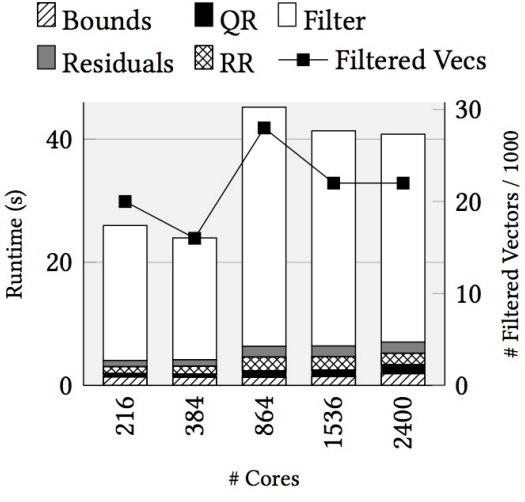
Table: Weak scaling experiment results

	Iterations		Matvecs		Runtime		
# Cores	ChASE-BLAS	ChASE-Elemental	ChASE-BLAS	ChASE-Elemental	ChASE-BLAS	ChASE-Elemental	Direct
216	11	11	19,990	20,192	25.1 s	26.0 s	81.5 s
384	10	9	16,778	16,100	23.7 s	24.0 s	141.2 s
864	17	11	23,424	27,506	39.8 s	45.2 s	211.1 s
1536	13	12	23,268	21,940	36.4 s	41.4 s	367.8 s
2400	10	13	22,614	21,720	38.4 s	40.8 s	380.1 s

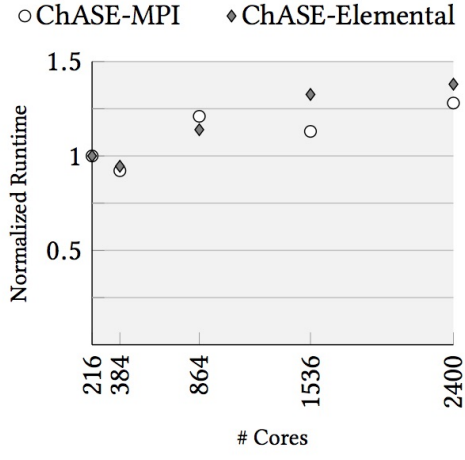
Tests were performed on the JURECA cluster.

- 2 Intel Xeon E5-2680 v3 Haswell – Up to $0.96 \div 1.92$ TFLOPS DP/SP;
- 2 x NVIDIA K80 (four devices) – Up to $2.91 \div 8.74$ TFLOPS DP/SP.

WEAK SCALING



(a) Timings in ChASE-Elemental's subroutines



(b) Normalized Runtimes

CONCLUSIONS AND OUTLOOK

ChASE library RELEASED.

- ChASE is open source (BSD 2.0 license) and available at
- `https://github.com/SimLabQuantumMaterials/ChASE`
- Submission to TOMS at the end of April 2018.

Future Work

- Jena BSE parallel I/O matrix generation
- Tuning ChASE for Blue Waters and the BSE application
- Large scale (weak) scaling on Blue Waters (with Hamiltonians up to 1,000,000
- Accuracy of Single Precision GPU `CGEMM`

Acknowledgments

- **Xiao Zhang (Oxide semiconductors)**
- **Josh Leveillee (Hybrid perovskites)**

CBET-1437230
DMR-1555153

OCI-0725070
ACI-1238993

ACKNOWLEDGEMENTS (CONT'D)

For more information

e.di.napoli@fz-juelich.de

fz-juelich.de/ias/jsc/slqm

schleife@illinois.edu

schleife.matse.illinois.edu

Special thanks to JLESC for providing travel funding.

Financial support from the Deutsche Forschungsgemeinschaft (German Research Association) through grant GSC 111 is gratefully acknowledged.