Efficient computation of sparse matrix functions for large scale electronic structure calculations: The CheSS library

Stephan Mohr, William Dawson, Michael Wagner, Damien Caliste, Takahito Nakajima, Luigi Genovese

Barcelona Supercomputing Center

MaX Conference

Trieste, 30th January 2018

Outline

Motivation for CheSS

Linear Algebra operations appear in all electronic structure codes and are in principle easy to handle (standard libraries).

However they can become a severe bottleneck (up to $\mathcal{O}(N^3)$).

The straightforward approach is wasteful in the case of sparse matrices:

- consequence of a localized basis set (e.g. Gaussians, wavelets, etc.)
- intrinsic localization properties of the system

We created a standalone library for sparse Linear Algebra operations, specifically tailored for electronic structure codes:
Chebyshev **S**parse **S**olvers (CheSS).

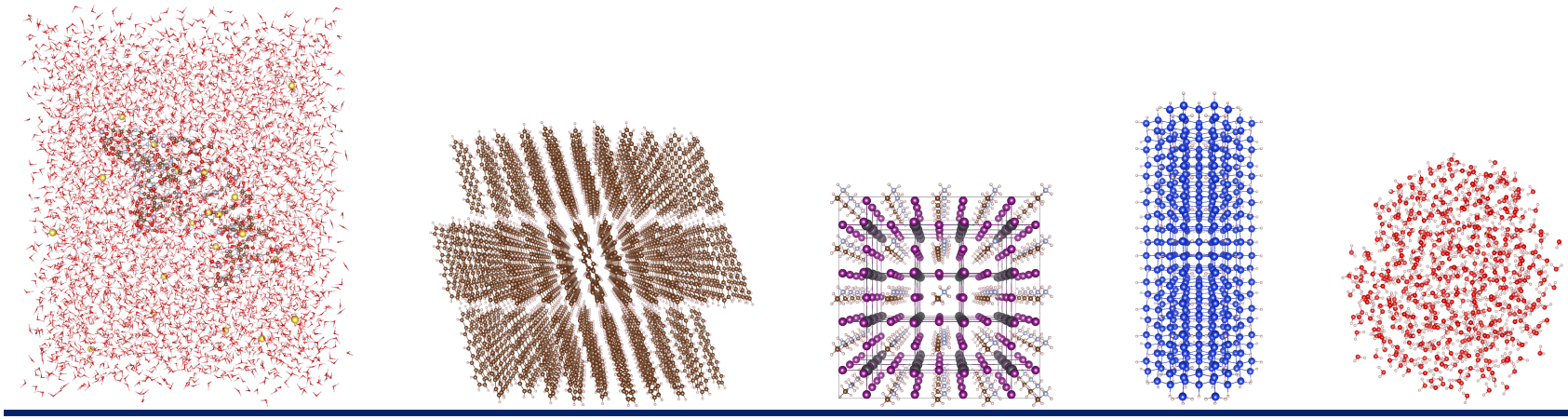
CheSS can be obtained for free from `https://launchpad.net/chess`

More details in J. Chem. Theory Comput., 2017, **13** (10), pp 4684–4698

Applicability of CheSS

CheSS performs best for matrices exhibiting a small spectral width.

Can this be obtained in practice?



		S				H				
system	#atoms	sparsity	ϵ_{min}	ϵ_{max}	κ	sparsity	ϵ_{min}	ϵ_{max}	λ	Δ_{HL}
DNA	15613	99.57%	0.72	1.65	2.29	98.46%	-29.58	19.67	49.25	2.76
bulk pentacene	6876	98.96%	0.78	1.77	2.26	97.11%	-21.83	20.47	42.30	1.03
perovskite	768	90.34%	0.70	1.50	2.15	76.47%	-20.41	26.85	47.25	2.19
Si nanowire	706	93.24%	0.72	1.54	2.16	81.61%	-16.03	25.50	41.54	2.29
water	1800	96.71%	0.83	1.30	1.57	90.06%	-26.55	11.71	38.26	9.95

Basic idea

In CheSS we approximate matrix functions by Chebyshev polynomials:

$$p(\mathbf{M}) = \frac{c_0}{2}\mathbf{I} + \sum_{i=1}^{n_{pl}} c_i \mathbf{T}^i(\tilde{\mathbf{M}}) \,,$$

with

$$\tilde{\mathbf{M}} = \sigma(\mathbf{M} - \tau\mathbf{I}) \quad ; \quad \sigma = \frac{2}{\epsilon_{max} - \epsilon_{min}} \quad ; \quad \tau = \frac{\epsilon_{min} + \epsilon_{max}}{2}$$

and

$$c_j = \frac{2}{n_{pl}} \sum_{k=0}^{n_{pl}-1} f\left[\frac{1}{\sigma}\cos\left(\frac{\pi(k+\frac{1}{2})}{n_{pl}}\right) + \tau\right] \cos\left(\frac{\pi j(k+\frac{1}{2}}{n_{pl}}\right) \,.$$

Recursion relation for the Chebyshev polynomials:

$$\mathbf{T}^0(\tilde{\mathbf{M}}) = \mathbf{I} \,,$$
$$\mathbf{T}^1(\tilde{\mathbf{M}}) = \tilde{\mathbf{M}} \,,$$
$$\mathbf{T}^{j+1}(\tilde{\mathbf{M}}) = 2\tilde{\mathbf{M}}\mathbf{T}^j(\tilde{\mathbf{M}}) - \mathbf{T}^{j-1}(\tilde{\mathbf{M}}) \,.$$

Each column independent
$\Longrightarrow$ easily parallelizable

Strict sparsity pattern
$\Longrightarrow$ linear scaling

Available functions

CheSS can calculate those matrix functions needed for DFT:

- density matrix: $f(x) = \frac{1}{1+e^{\beta(x-\mu)}}$
  (or $f(x) = \frac{1}{2}\left[1 - \mathrm{erf}\left(\beta(\epsilon - x)\right)\right]$)
- energy density matrix: $f(x) = \frac{x}{1+e^{\beta(x-\mu)}}$
  (or $f(x) = \frac{x}{2}\left[1 - \mathrm{erf}\left(\beta(\epsilon - x)\right)\right]$)
- matrix powers: $f(x) = x^a$ (a can be non-integer!)

We can calculate arbitrary functions by changing only the coefficients c_j!

Only requirement:
function f must be well representable by Chebyshev polynomials over the entire eigenvalue spectrum.

Sparsity and truncation

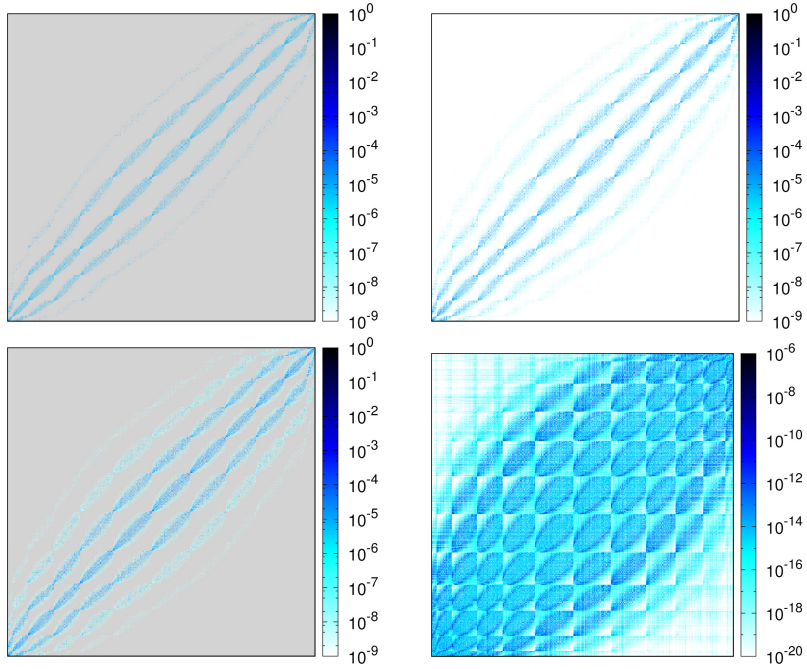
CheSS works with predefined sparsity patterns.

In general there are three:

- pattern for the original matrix $\mathbf{M}$
- pattern for the matrix function $f(\mathbf{M})$
- auxiliary pattern to perform the matrix-vector multiplications

At the moment all of them must be defined by the user.
Typically: distances atoms / basis functions

1. original matrix $\mathbf{M}$
2. exact calculation of $\mathbf{M}^{-1}$ without sparsity constraints
3. sparse calculation of $\mathbf{M}^{-1}$ using CheSS within the sparsity pattern
4. difference between Fig. 2 and Fig. 3



Accuracy – error definition

There are two possible factors affecting the accuracy of CheSS:

- error introduced due to the enforced sparsity (truncating the matrix-vector multiplications)
- error introduced by the Chebyshev fit

This also affects the definition of the "exact solution". Two possibilities:

1. calculate the solution exactly and without sparsity constraints and then crop to the sparsity pattern. Shortcoming: violates in general the identity $f^{-1}(f(\mathbf{M})) = \mathbf{M}$

   Associated error: $w_{\hat{f}_{sparse}} = \frac{1}{|\hat{f}(\mathbf{M})|}\sqrt{\sum_{(\alpha\beta)\in\hat{f}(\mathbf{M})}\left(\hat{f}(\mathbf{M})_{\alpha\beta} - f(\mathbf{M})_{\alpha\beta}\right)^2}$

2. calculate the solution within the sparsity pattern, and define as exact one that which fulfills $\hat{f}^{-1}(\hat{f}(\mathbf{M})) = \mathbf{M}$

   Associated error: $w_{\hat{f}^{-1}(\hat{f})} = \frac{1}{|\hat{f}(\mathbf{M})|}\sqrt{\sum_{(\alpha\beta)\in\hat{f}(\mathbf{M})}\left(\hat{f}^{-1}(\hat{f}(\mathbf{M}))_{\alpha\beta} - M_{\alpha\beta}\right)^2}$

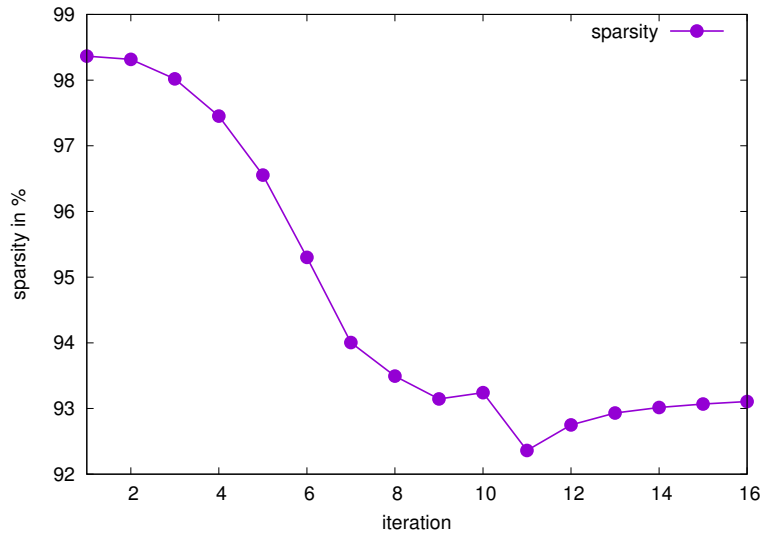
Fixed versus variable sparsity

Alternative to a fixed sparsity pattern: Variable adaption during the iterations (e.g. based on the magnitude of the entries).

- Advantage: Flexible control of the truncation error
- Shortcoming: The sparsity may decrease considerably

Example: Purification method (TRS4 by Niklasson)

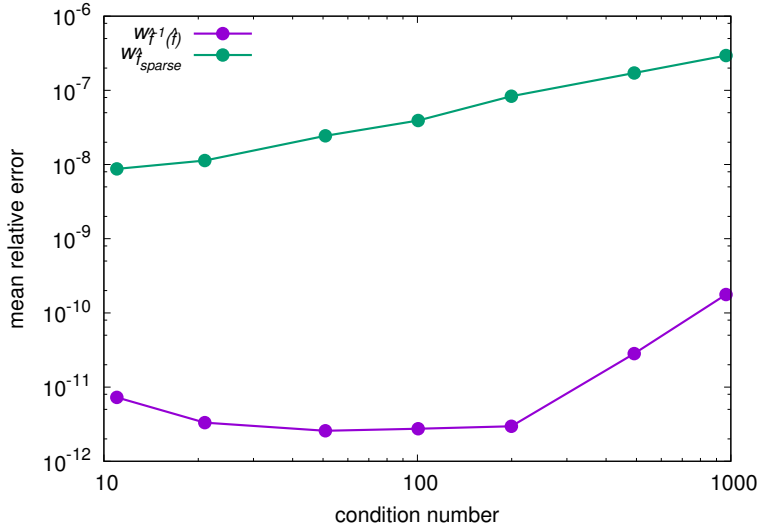
- Sparsity defined by magnitude
- Strong fill-in during the iterations
- Cost of matrix multiplications increases
- Unlike CheSS the matrix to be applied changes $\Rightarrow$ harder to parallelize



Accuracy

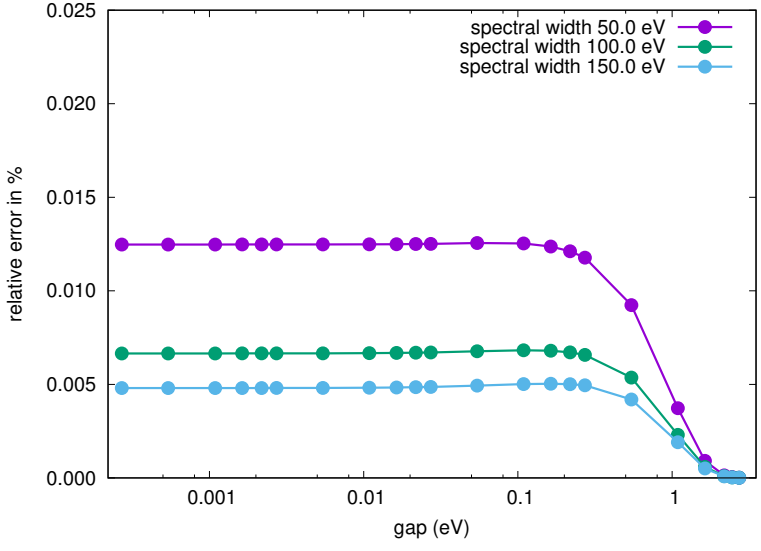
Inverse:

- $w_{\hat{f}^{-1}(\hat{f})}$: error due to Chebyshev fit basically zero
- $w_{\hat{f}_{sparse}}$: error due to sparsity pattern very small



Density matrix:

- energy (i.e. Tr(**KH**)): relative error of only 0.01%
- slightly larger error for small spectral width:
  eigenvalues are denser, finite temperature smearing affects more

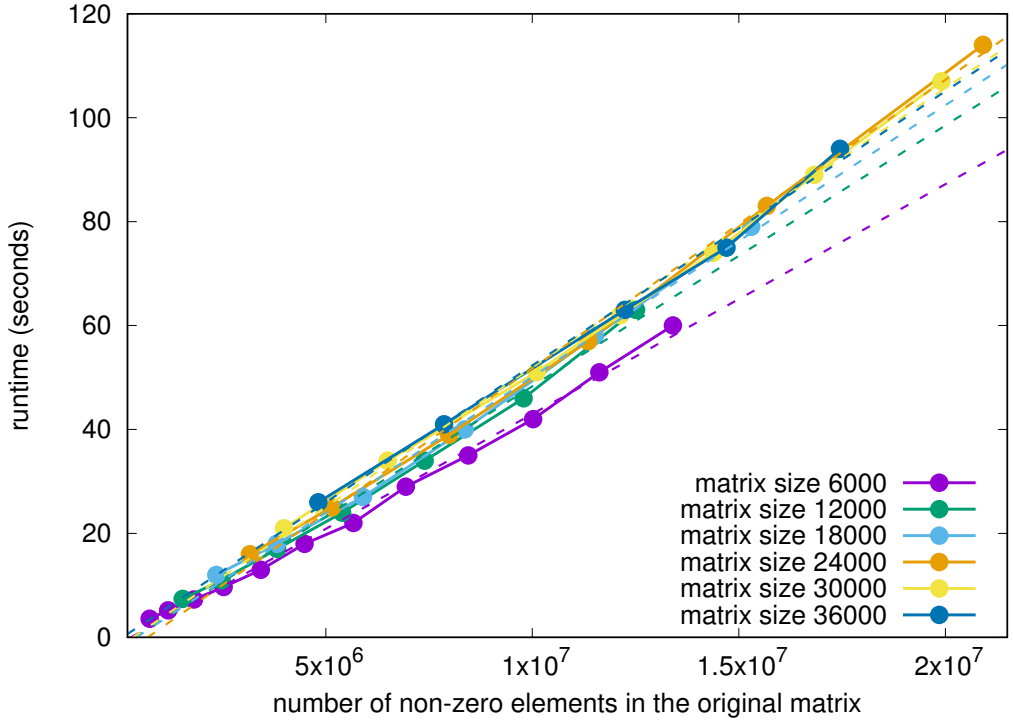


Scaling with matrix size and sparsity

Series of matrices with the same "degree of sparsity"
(DFT calculations of water droplet of various size).

Example: calculation of the inverse

- Runtime only depends on the number of non-zero elements of **M**
- no dependence on the total matrix size

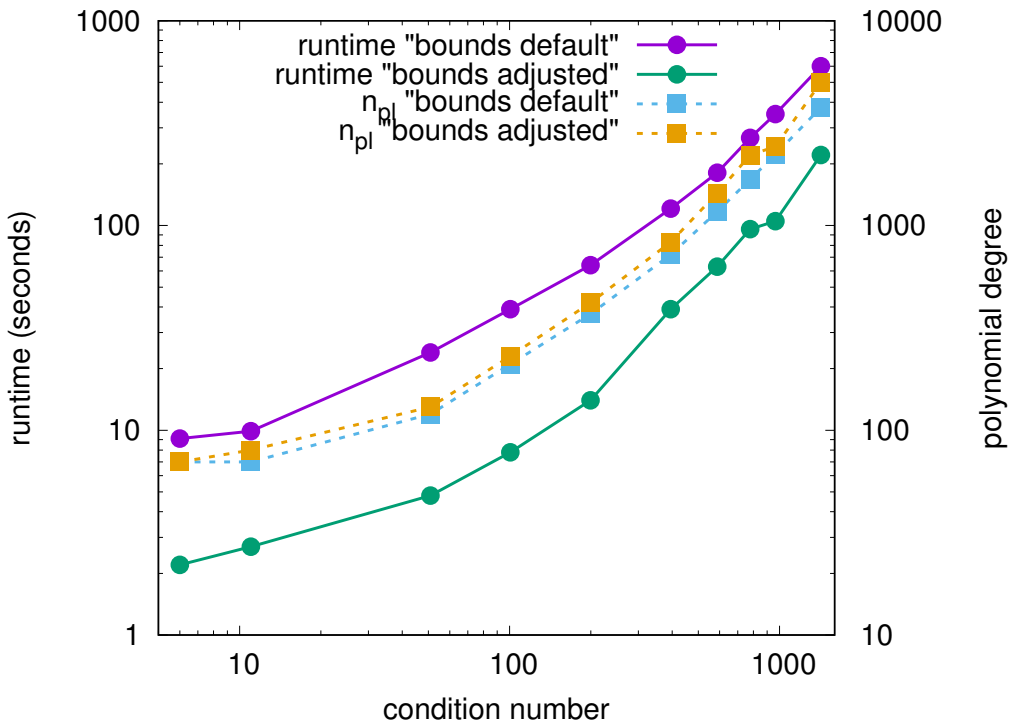


Scaling with spectral properties

CheSS is extremely sensitive to the eigenvalue spectrum:

- required polynomial degree strongly increases with the spectral width
- as a consequence the runtime strongly increases as well
- a good input guess for the eigenvalue bounds helps a lot

Example: calculation of the inverse

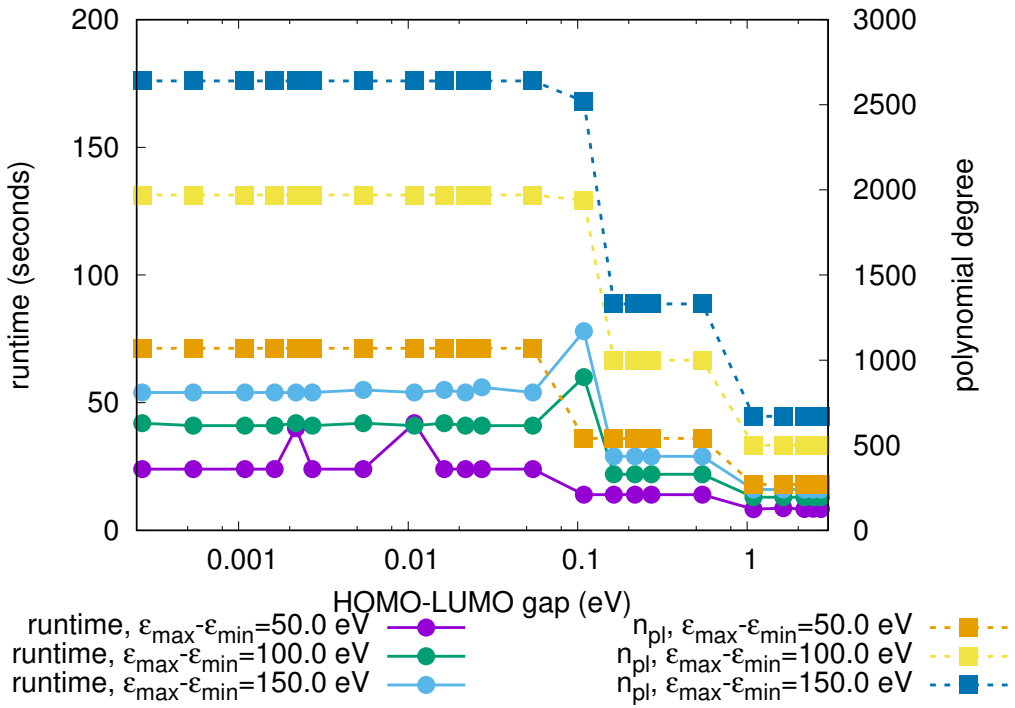


Scaling with spectral properties

For the density matrix the performance depends on two parameters:

- spectral width (the smaller the better)
- HOMO-LUMO gap (the larger the better)

In both cases the polynomial degree can increase considerably
$\Longrightarrow$ CheSS less efficient.

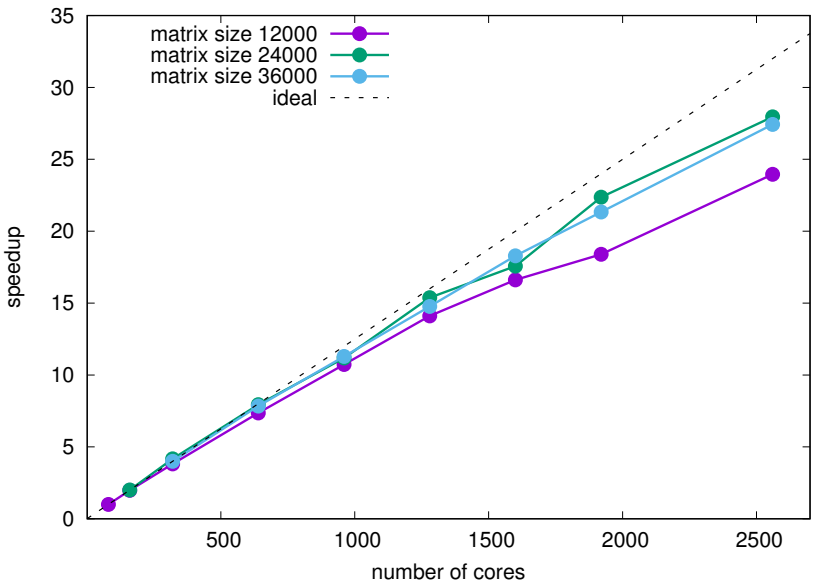


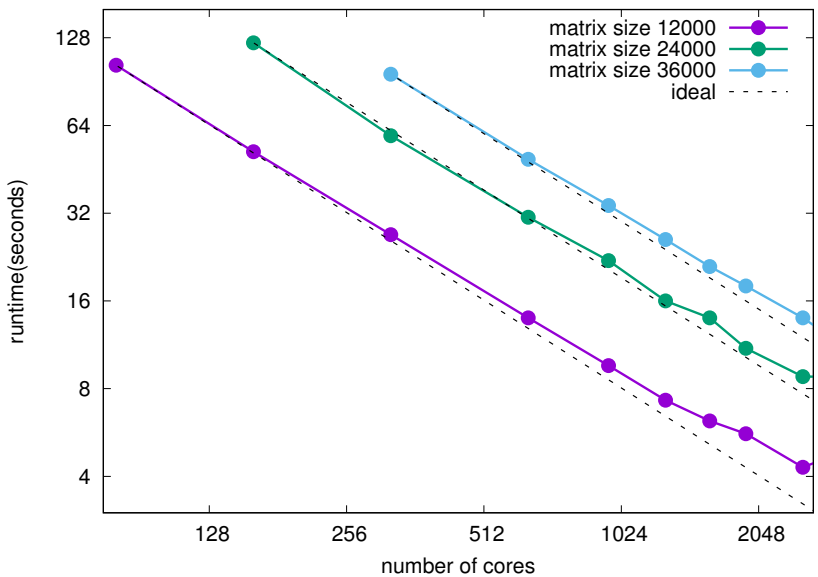
Parallel scaling

Most compute intensive part of CheSS: matrix-vector multiplications.

- Easily parallelizable
- identical for all operations

Example: Calculation of $\mathbf{M}^{-1}$ (runs performed with 16 OpenMP threads)

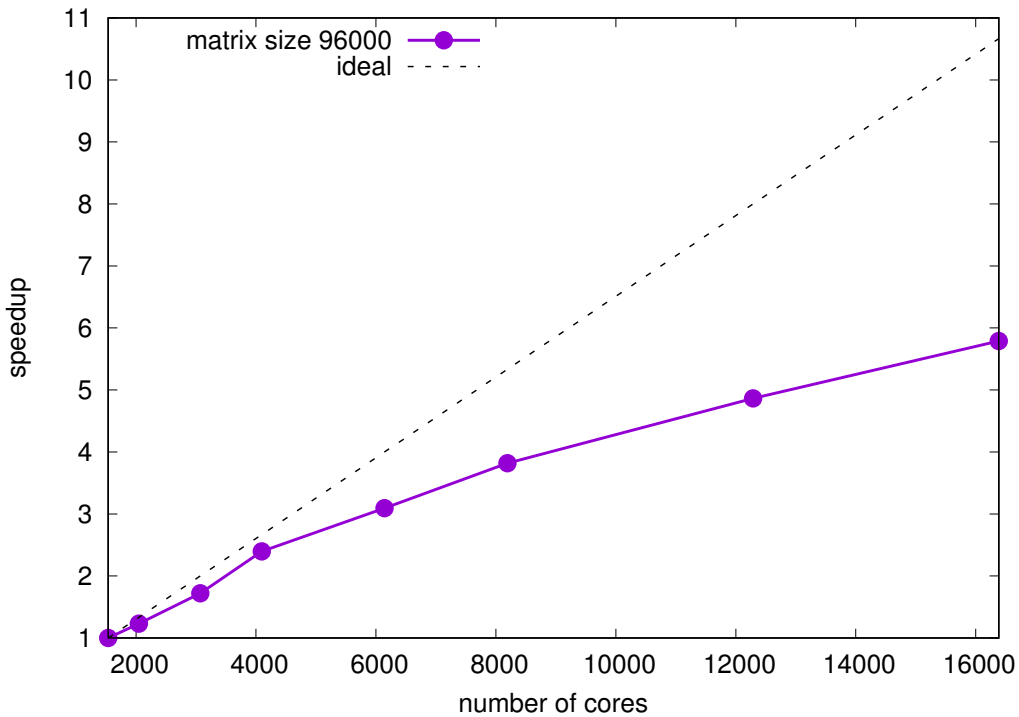




Extreme scaling

We have also performed extreme-scaling tests from 1536 to 16384 cores.

Example: Calculation of $\mathbf{M}^{-1}$ (runs performed with 8 OpenMP threads)



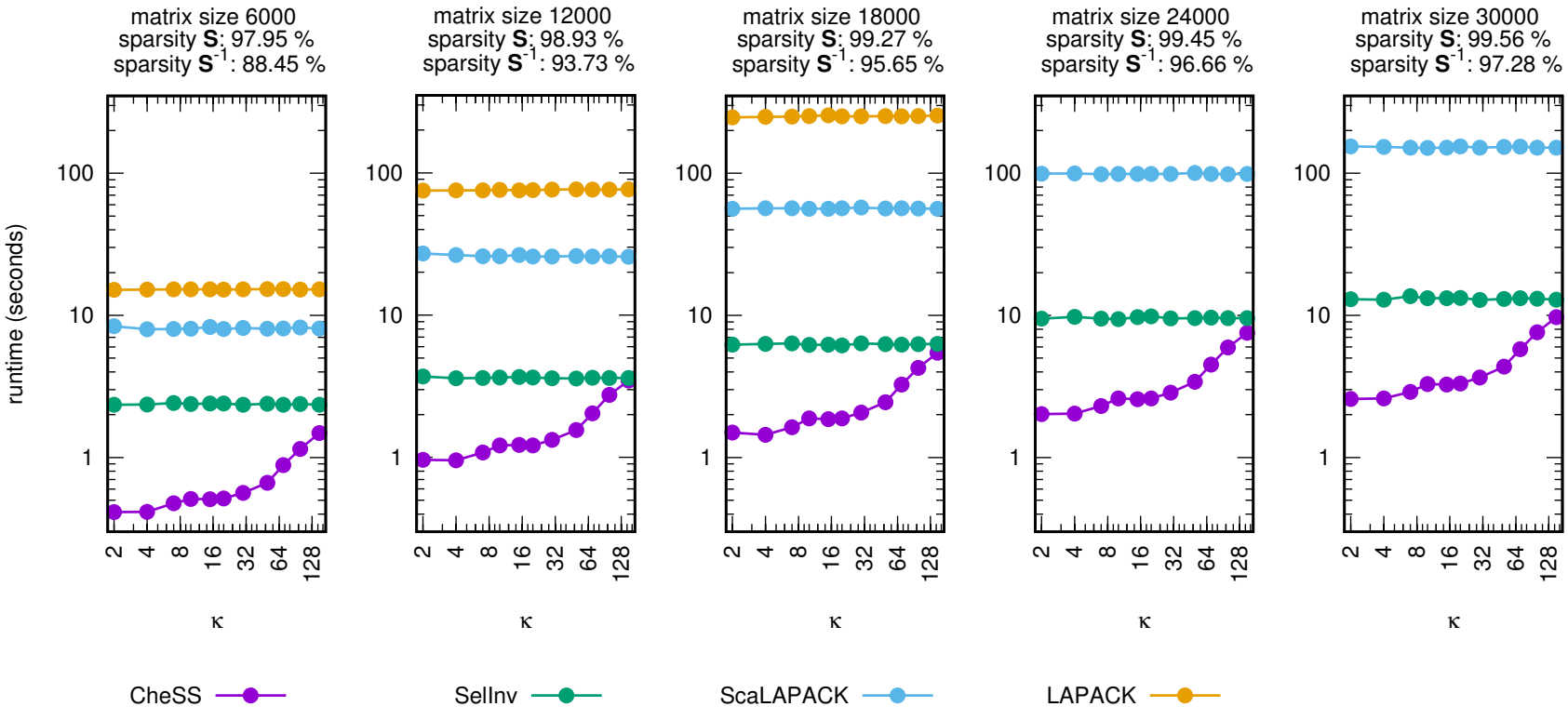
Given the small matrix (96000×96000) the obtained scaling is good.

We will try to further improve the scaling.

Comparison with other methods: Inverse

Comparison of the matrix inversion between:

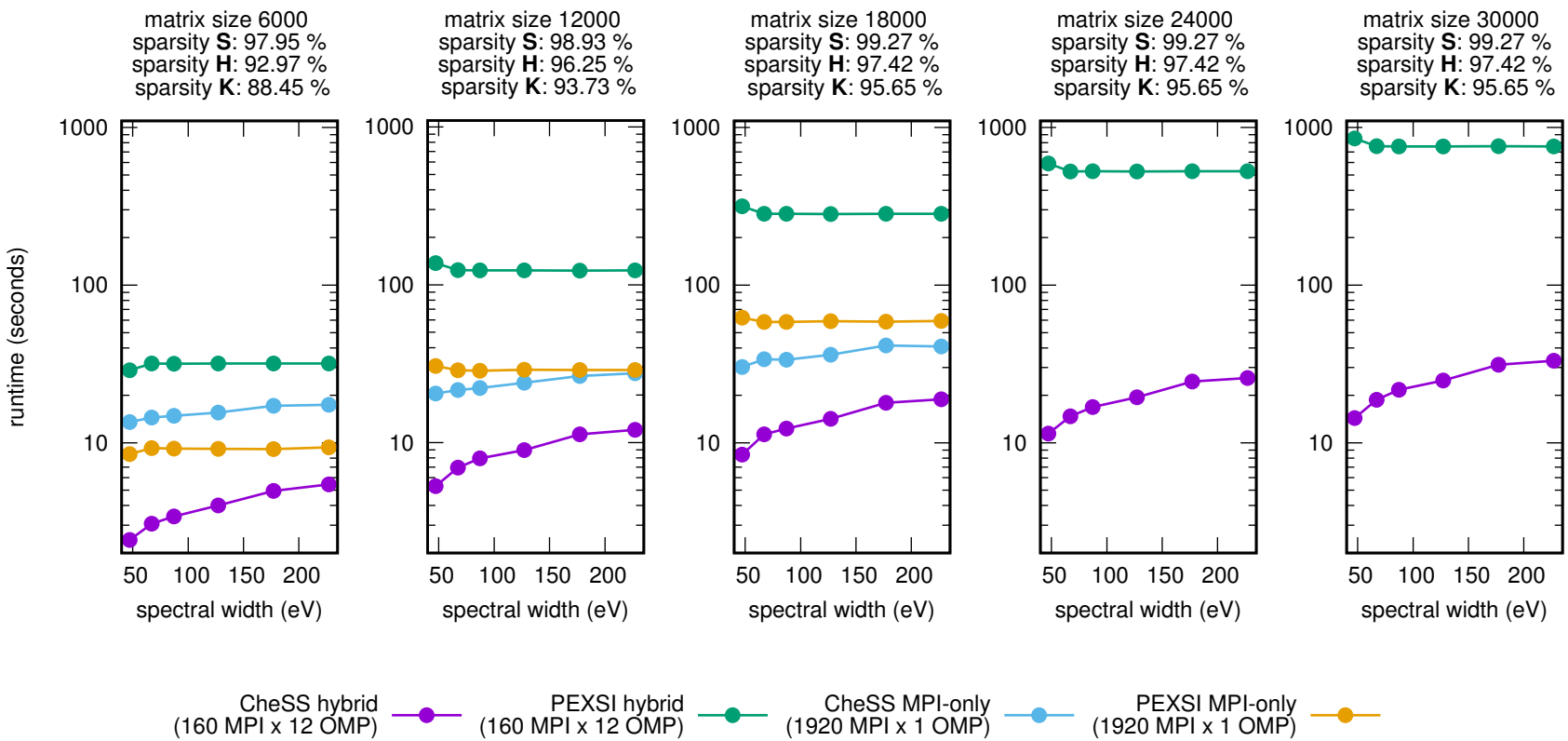
- CheSS
- SelInv
- ScaLAPACK
- LAPACK



Comparison with other methods: Density matrix

Comparison of the density matrix calculation between:

- CheSS hybrid MPI/OpenMP
- PEXSI hybrid MPI/OpenMP
- CheSS MPI-only
- PEXSI MPI-only



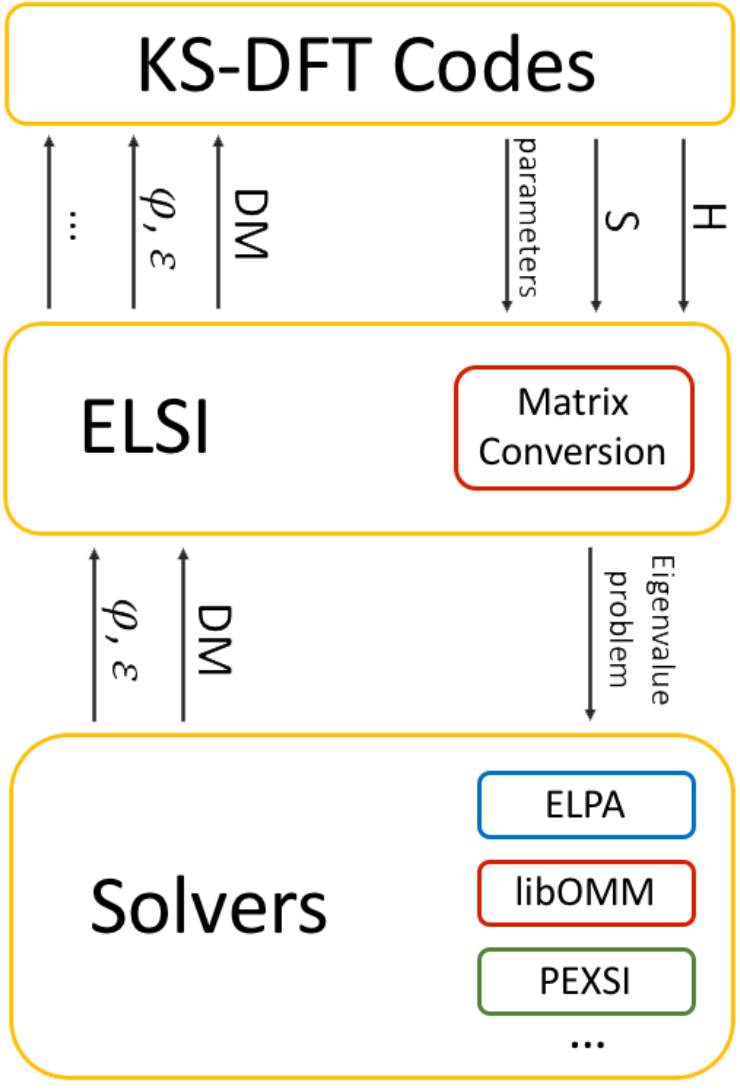
Integration into ELSI

There are various other libraries to accelerate linear algebra operations, e.g.

- ELPA
- libOMM
- PEXSI
- ...

The ELSI project tries to provide one single interface for all these libraries.

We are planning to include also CheSS in ELSI.



Conclusions

- CheSS is a flexible tool to calculate matrix functions for DFT
- can easily be extended to other functions
- exploits sparsity of the matrices, linear scaling possible
- works best for small spectral widths of the matrices
- very good parallel scaling (both MPI and OpenMP)

Thank you for your attention!

stephan.mohr@bsc.es