Interdisciplinary software for electronic structure theory: adcc and DFTK

Michael F. Herbst

Matherials team, CERMICS, Ecole des Ponts ParisTech

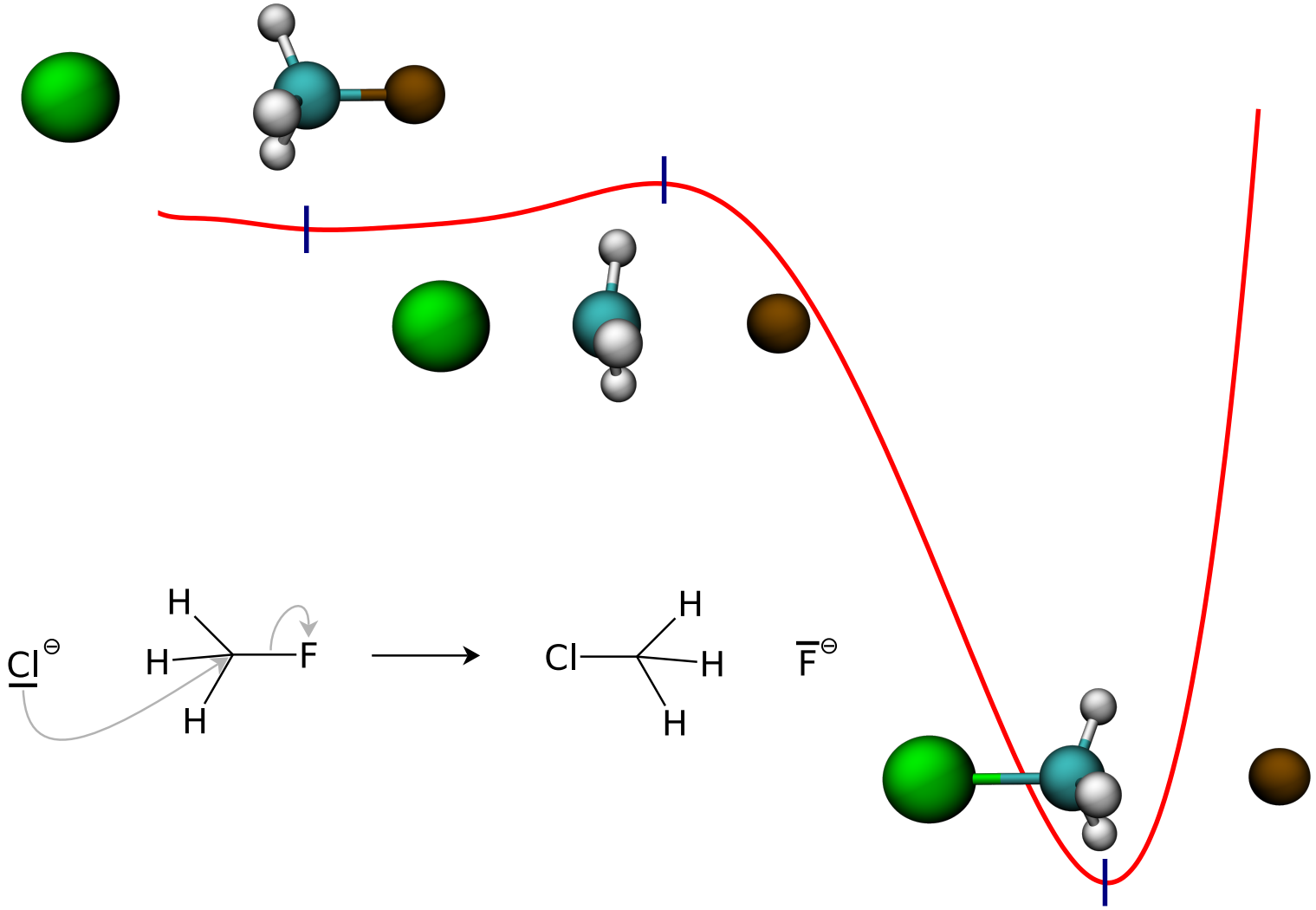
11th September 2019

https://michael-herbst.com/talks/2019.09.11_adcc_dftk_munich.pdf

Contents

Contents

Describing chemistry



Solving the Schrödinger equation: How hard can it be?

- Main ingredient: Min-max (variational) principle

$$E_0 \leq \min_{\Psi} \mathcal{E}(\Psi) = \min_{\Psi} \frac{\left\langle \Psi \middle| \hat{\mathcal{H}} \Psi \right\rangle}{\langle \Psi | \Psi \rangle}$$

- Discretisation: Curse of dimensionality:
  - $\langle \cdot | \cdot \rangle$ involves integral over $3N$-dim. space
  - Assume 2 quadrature points only
  - Chloromethane: $N = 26 \Rightarrow 2^{78} \approx 3 \cdot 10^{23}$ quadrature points

  $\Rightarrow$ Finished in 1 year:

Solving the Schrödinger equation: How hard can it be?

- Main ingredient: Min-max (variational) principle

$$E_0 \leq \min_{\Psi} \mathcal{E}(\Psi) = \min_{\Psi} \frac{\left\langle \Psi \middle| \hat{\mathcal{H}} \Psi \right\rangle}{\langle \Psi | \Psi \rangle}$$

- Discretisation: Curse of dimensionality:
  - $\langle \cdot | \cdot \rangle$ involves integral over $3N$-dim. space
  - Assume 2 quadrature points only
  - Chloromethane: $N = 26 \Rightarrow 2^{78} \approx 3 \cdot 10^{23}$ quadrature points

$\Rightarrow$ Finished in 1 year: ≈ 100 attoseconds per quadrature point

Now what?

- Starting point: Single determinant theories (HF, DFT)
  - Slater determinant $|\Psi\rangle = |\psi_0 \psi_1 \cdots \psi_N\rangle$
  - Effective one-particle operator $\hat{\mathcal{F}}$
- Discretise $\hat{\mathcal{F}}$ in a basis $\{\varphi_\mu\}_\mu$:

$$F_{\mu\nu} = \left\langle \varphi_\mu \middle| \hat{\mathcal{F}} \varphi_\nu \right\rangle$$
$$C_{\mu,i} = \langle \varphi_\mu | \psi_i \rangle$$

- Discretised self-consistent problem:

$$\mathbf{F}[\mathbf{C}]\,\mathbf{C} = \mathbf{S}\mathbf{C}\ \operatorname{diag}(\varepsilon_1, \ldots, \varepsilon_n)$$

Some mathematically motivated questions

- Does the exact problem have a well-defined answer?
- Does the discretised problem yield the same answer?
- How far off is it? How can we improve best?
- How to construct a better discretisation basis?

- Balance of accuracy between method and numerics?
- Agreement of model / numerics with computational hardware?

$\Rightarrow$ Interdisciplinary efforts

But is this even relevant?

- Consider high-throughput screening
  - Drug design, catalysis, material science, ...
- Need reliable black-box codes
- As much speed as possible to get target accuracy
  - ⇒ Error bounds
  - ⇒ Rigorous numerics
  - ⇒ Leverage available hardware
- So: Can we tackle such questions?

Mathematicians start small . . .

- Simplified / asymptotic problems
- Often barely related to physical models
- Existence / uniqueness / optimality of solutions
- From insight: New numerics / algorithms
  - E.g. Taylor series and complex residuals

⇒ Often need numerical experiments

⇒ A new code base for each toy problem?

. . . but we want high-performance!

- Compute clusters become increasingly heterogeneous:
  - Multi-core CPUs and multi-CPU nodes
  - Accelerators: GPUs, FPGAs, . . .
- High-performance architecture constantly changes
- Strength and weaknesses differ

- Impact on all levels:
  - Approximate model
  - Discretisation / basis functions
  - Algorithmic approach to numerics

$\Rightarrow$ Rewrite code from scratch for each architecture?

Demands for interdisciplinary software

- **Mathematicians:** Toy models and unphysical edge cases
- **Scientist:** Wants to focus on science, not numerics
- **High-performance:** Exploit all hardware specialities
- **Practitioner:**
  - Reliable, black-box, high-level for setup and data analysis

- Everything in one project?
- Need good compromise and suitable programming language
- Two of my tries: `adcc` and `DFTK`

Contents

Plane-wave DFT in one slide

- Periodic and solid-state problems
- Plane-wave discretisations:

$$\forall \underline{\boldsymbol{k}} : \ \varphi_{\underline{\boldsymbol{G}}} = e^{\imath \underline{\boldsymbol{k}} \cdot \underline{\boldsymbol{x}}} \, e^{\imath \underline{\boldsymbol{G}} \cdot \underline{\boldsymbol{x}}} \qquad G^2 < 2E_{\text{cut}}$$

- All-electron calculations:
  - Near nuclei: Core orbitals are sharp, valence orbitals oscillate

  $\Rightarrow$ Large E_{cut}

- Solution: Pseudopotentials
  - Remove core electrons
  - Smoothen valence orbitals

Some questions related to PW-DFT

- What's the effect of the pseudopotentials?
- Elevated / reduced precision?
- Support multiple accelerators in one code base?
- Error estimates / mixed grid methods?
- A fast and reliable SCF algorithm?

The approach of DFTK

- DFTK: density-functional toolkit:
  - Minimalistic code base (not a program package)
  - Use existing libraries and codes
  - Facilitate integration elsewhere
  - Joint with Antoune Levitt and Eric Cancès
- Accessible to mathematicians, physicists, computer scientists
  - Use custom Hamiltonians, potentials
  - Build new models on a high-level
  - Target modern HPC environments

⇒ Toy problems and full-scale applications

DFTK is written in julia

- High-level, dynamical language for HPC:
  - Parallelism, vectorisation, GPU, automatic differentiation, ...
- Key concept: Multiple dispatch
- At runtime: Function compiled *exactly* for argument types
  - ⇒ Easy parallelisation and vectorisation
  - ⇒ Type-specific and hardware-specific optimisations
  - ⇒ E.g. allows to switch computational back end
- Write code once, re-use for many back ends / machines ...
- Interoperability: `FORTRAN`, `C`, `C++`, `python`, R, ...

≈ `python` with deeply integrated numpy

A word about julia performance

		duration (s)
python	array operations (numpy)	11.8
C	gcc	8.1
C	gcc -O3	1.1
C	gcc -O3 -march=native	**0.5**
FORTRAN	gfortran -O3 -march=native	**0.5**
julia	array operations	3.3
julia	loops	1.9
julia	loops, no bounds check	**0.5**

- Best out of five on my laptop (C, Julia, python code: Antoine Levitt)
- Used software: gcc 8.3, gfortran 9.2.1, python 3.7, numpy 1.16.2, julia 1.0.3

DEMO

DEMO

Show-casing `julia`

Status of DFTK

- https://github.com/mfherbst/DFTK.jl
- LDA, GGA functionals from libxc, analytic potentials
- Multiple SCF algorithms
- Insulators and metals (smearing)
- Laptop-level parallelism
- Single and double precision

Contents

ADC in one slide

- Algebraic-diagrammatic construction (ADC) approach to electronic excitations
- Post-HF: Builds on Møller-Plesset PT ground states
- Intermediate states

$$|\Psi_n\rangle = \sum_I X_{I,n} \left|\tilde{\Psi}_I\right\rangle$$

- Hermitian eigenvalue problem

$$\mathbf{M}\mathbf{X} = \mathbf{\Omega}\mathbf{X}, \qquad \mathbf{X}^\dagger\mathbf{X} = \mathbf{I},$$

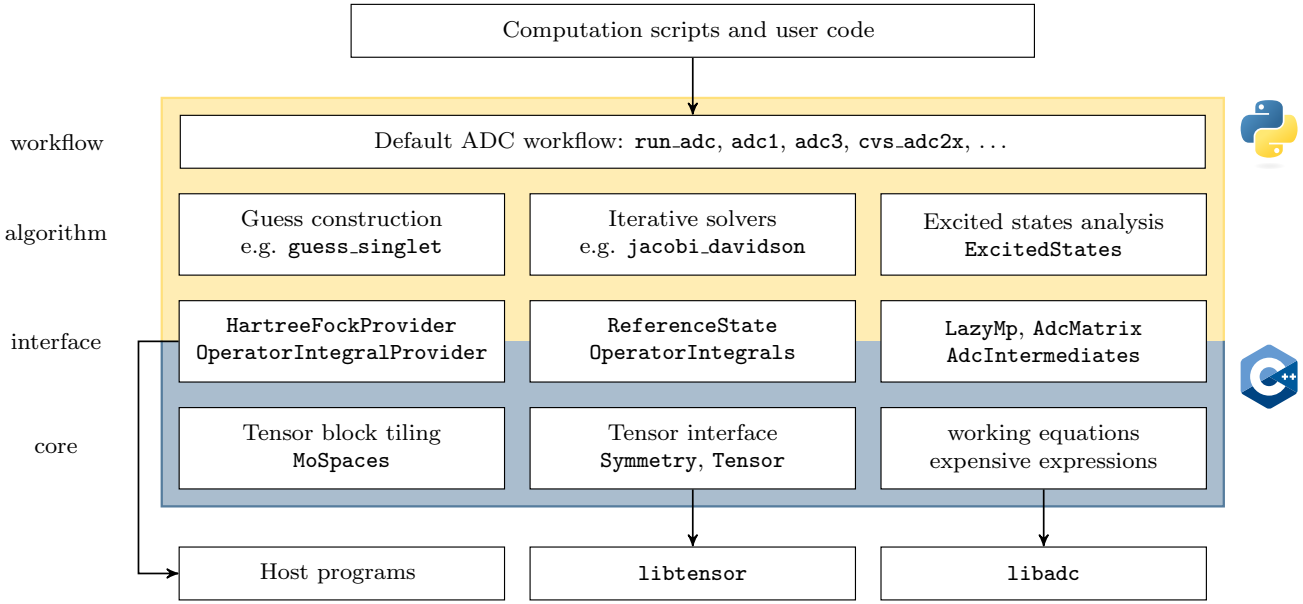
  with $\mathbf{M}$ ADC matrix and $\mathbf{\Omega}$ excitation energies.
- $\mathbf{M}$ sparse, so iterative methods employed (Jacobi-Davidson)

Some questions related to ADC

- ADC exists for multiple variants:
  - Core-valence separation (CVS)
  - Spin-flip
  - Frozen-core (FC) / frozen-virtual (FV)
- More specific numerics?
  - LOBPCG, Schur complement, preconditioning?
- Errors of CVS, FC, FV? Can these be undone?
- Interpolating from ADC(n) to ADC($n+1$)

$\Rightarrow$ Difficult to address in current frameworks

adcc:[1] Python-driven ADC for SCF codes



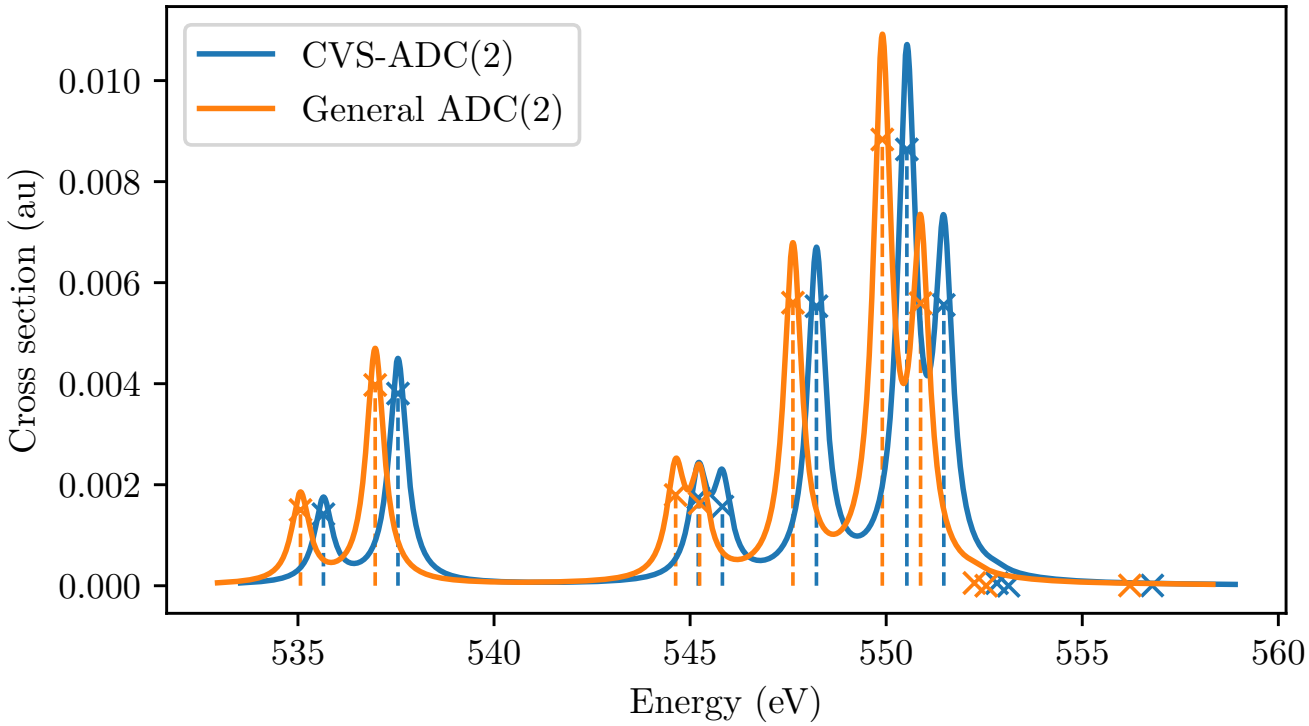
- Collaboration with Dreuw group, Heidelberg University

⇒ Simplified ADC development (numerics & methods)

⇒ Scales to medium-sized regime (≈ 450 basis functions)

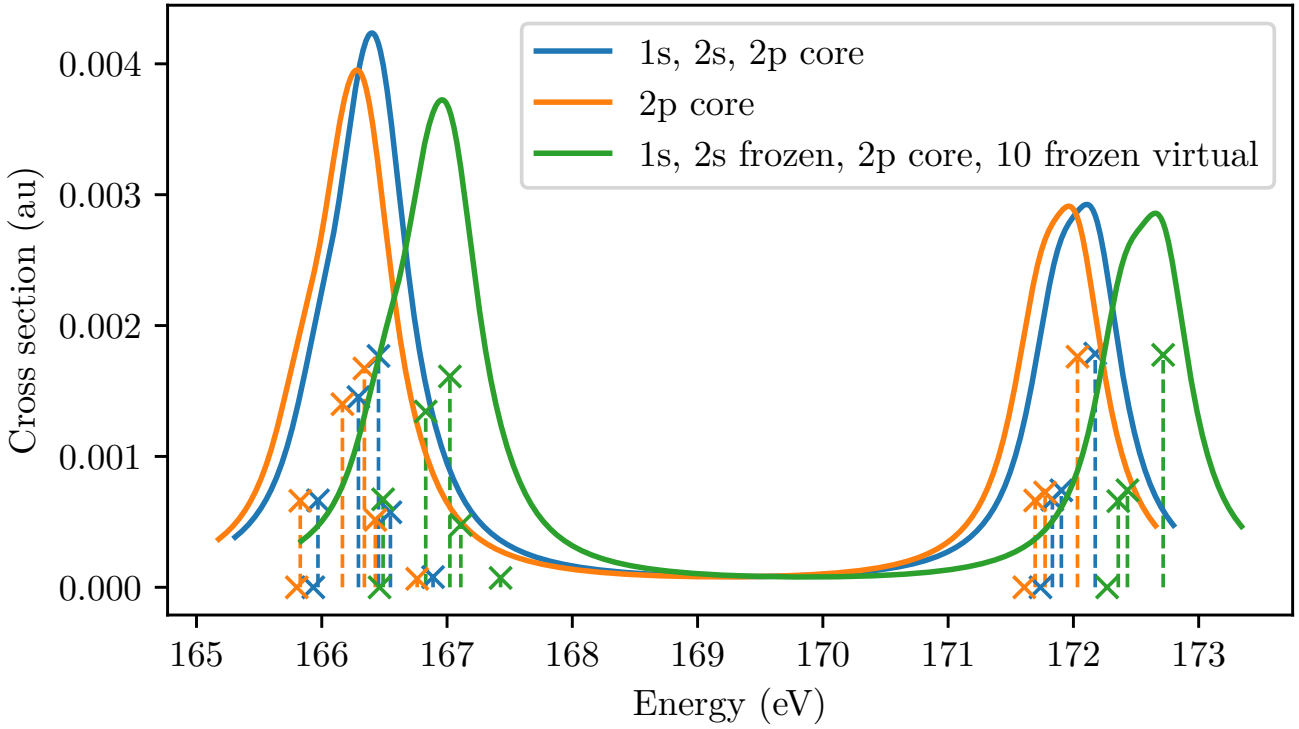
[1] Michael F. Herbst, Maximilian Scheurer *et. al.* *adcc: A versatile tool box for rapid-development of algebraic-diagrammatic construction methods*. In preparation

Example: The CVS error of water



- CVS-ADC(2), cc-pVTZ
- Relaxed by power iteration to full ADC(2) level
- CVS error: $0.6 \pm 0.01\,\mathrm{eV}$ (in this example)

Example: Tackling H_2S 2p core excitations

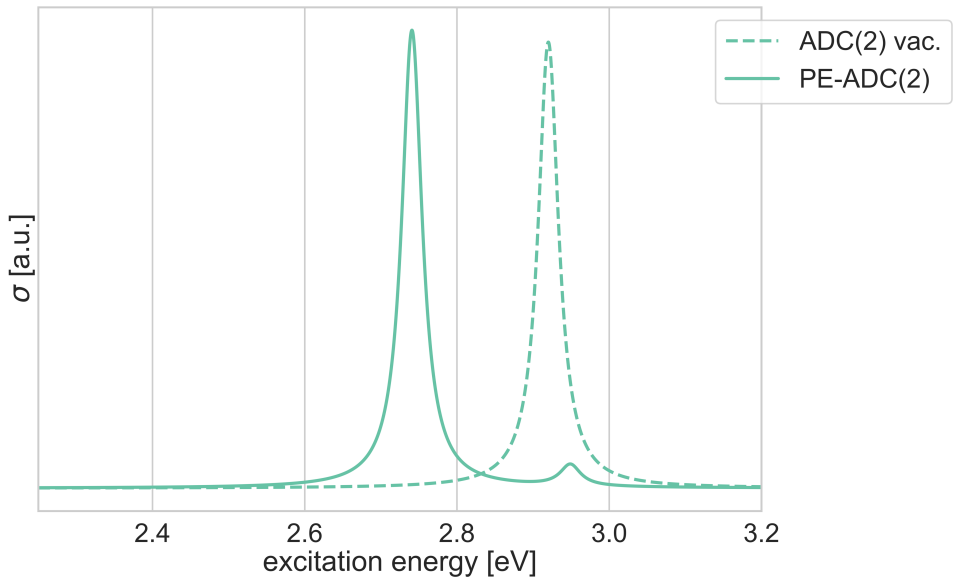


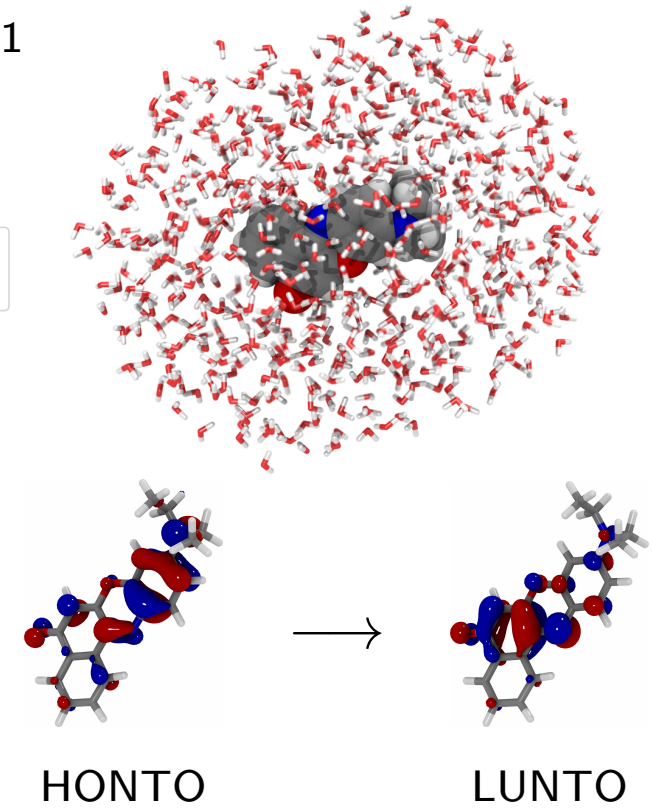
- cc-pVTZ, CVS-ADC(2)-x: 82 s, FC-FV-CVS-ADC(2)-x: 41 s
- FC-FV error about $0.5\,\mathrm{eV}$

⇒ Push the boundaries for CVS?

Example: Water solvent shift of nile red (PE)

- cc-pVDZ ADC(2), 426 bfctns, 2 states, 4.8h
- Polarisable embedding using CPPE[1]





[1] M. Scheurer, P. Reinholdt, *et. al.* *CPPE: An open-source C++ and Python library for Polarisable Embedding.* DOI 10.26434/chemrxiv.8949101.v1

DEMO

DEMO

Running excited-states calculations in adcc

Supported features

- Full python-driven ADC package
- CVS, FC, FV, spin-flip variants (all combinations)
- Multiple numerical schemes in python:
  - Jacobi-Davidson, conjugate-gradient, power iteration
- Performance comparable to C++-only implementation
- Supported host programs: Pyscf[1], Psi4[2], molsturm[3], veloxchem[4]
- Integration with further third-party codes, e.g. CPPE[5]

[1] Q. Sun,T. C. Berkelbach, *et. al.* WIREs Comput Mol Sci, **8**, e1340 (2017)

[2] R. M. Parrish, L. A. Burns, *et. al.* JCTC, **13**, 3185 (2017)

[3] M. F. Herbst, A. Dreuw and J. E. Avery. J. Chem. Phys., **149**, 84106 (2018)

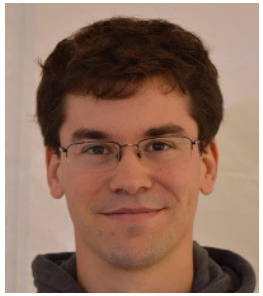
[4] Z. Rinkevicius, X. Li, *et. al. VeloxChem: A python-driven DFT program for spectroscopy simulations in HPC environments* (2019), *In preparation*.

[5] M. Scheurer, P. Reinholdt, *et. al. CPPE: An open-source C++ and Python library for Polarisable Embedding.*

Summary: DFTK and adcc

- Interdiciplinary software development
  - Aim: Achieve high-performance / reliability
  - No single, best choice of algorithms, discretisations, methods
  - Need high-level, rapid prototyping approach
- DFTK: Periodic problems
  - `julia` toolkit for different communities
  - Multi-precision, numerical analysis, HPC, new methods
  - Toy problems *and* full-scale applications
- adcc: Molecular spectroscopy
  - Feature-rich, ready-to-use ADC package (`python`/ `C++`)
  - Allows to explore new numerical approaches
  - Building block for ADC method development

Acknowledgements



Antoine Levitt

Eric Cancès

Maximilian Scheurer

Thomas Fransson

Adrian Dempwolff

Dirk Rehn

Andreas Dreuw

Questions?

DFTK: `https://github.com/mfherbst/DFTK.jl`

adcc: M. F. Herbst, M. Scheurer et al. *adcc: A versatile tool box for rapid-development of algebraic-diagrammatic construction methods* (2019).

In preparation

`https://adc-connect.org`

Email: `michael.herbst@enpc.fr`

Blog: `https://michael-herbst.com/blog`

Total time cost

- Total time cost is
  - Runtime of the production calculation

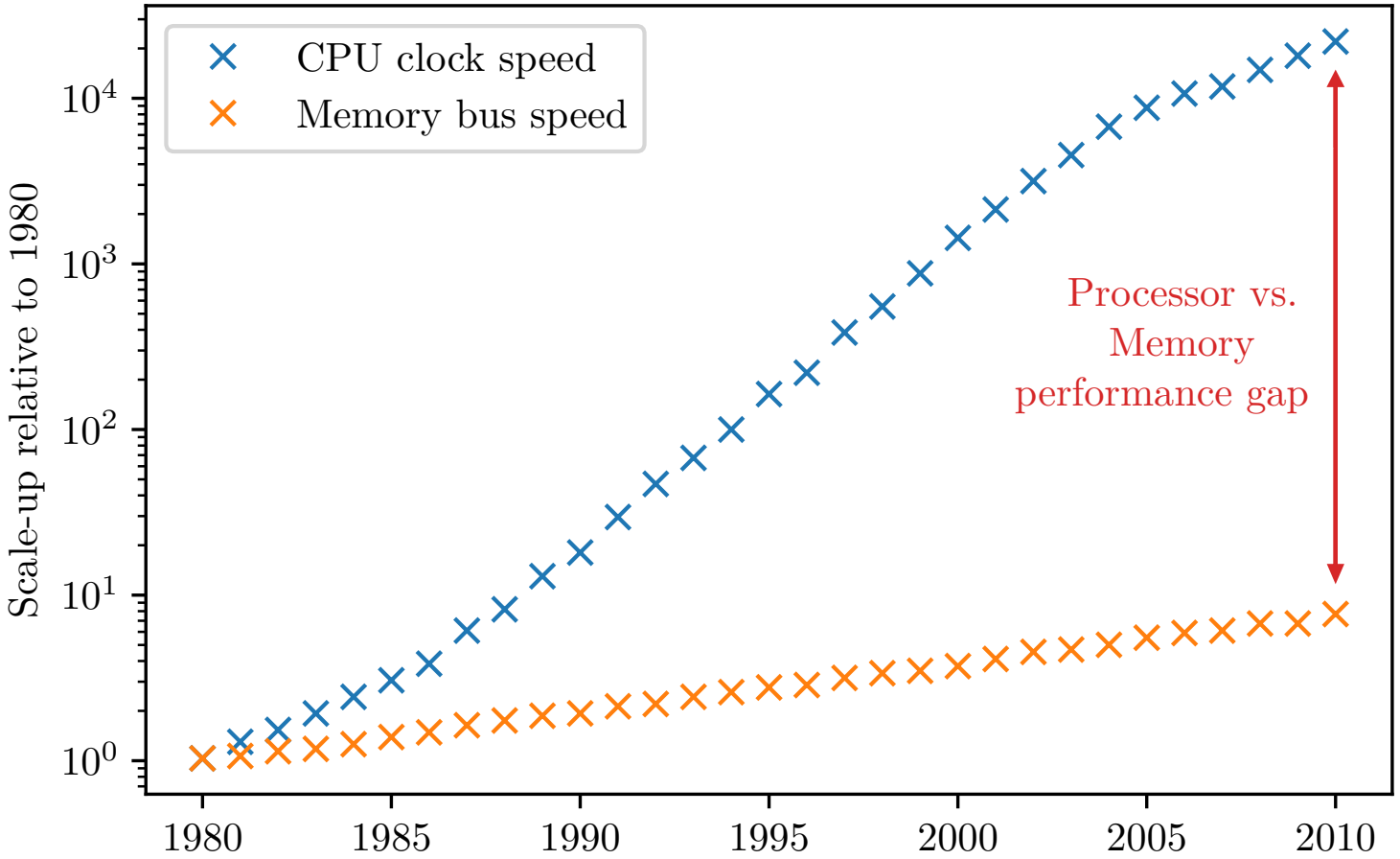
Total time cost

- Total time cost is sum of time for:
  - Understanding existing code
  - Implementing the new feature
  - Finding and fixing all the bugs
  - Optimising code performance
  - Runtime of the production calculation
  - Analysing and visualising results
  - Maintaining the code over its lifetime

Total time cost

- Total time cost is sum of time for:
  - Understanding existing code
  - Implementing the new feature
  - Finding and fixing all the bugs
  - Optimising code performance
  - Runtime of the production calculation
  - Analysing and visualising results
  - Maintaining the code over its lifetime

Memory versus processor developments



Data from `https://dave.cheney.net/2014/06/07/five-things-that-make-go-fast`