Modern software-development techniques in electronic structure theory

Michael F. Herbst

Matherials team, CERMICS, Ecole des Ponts ParisTech

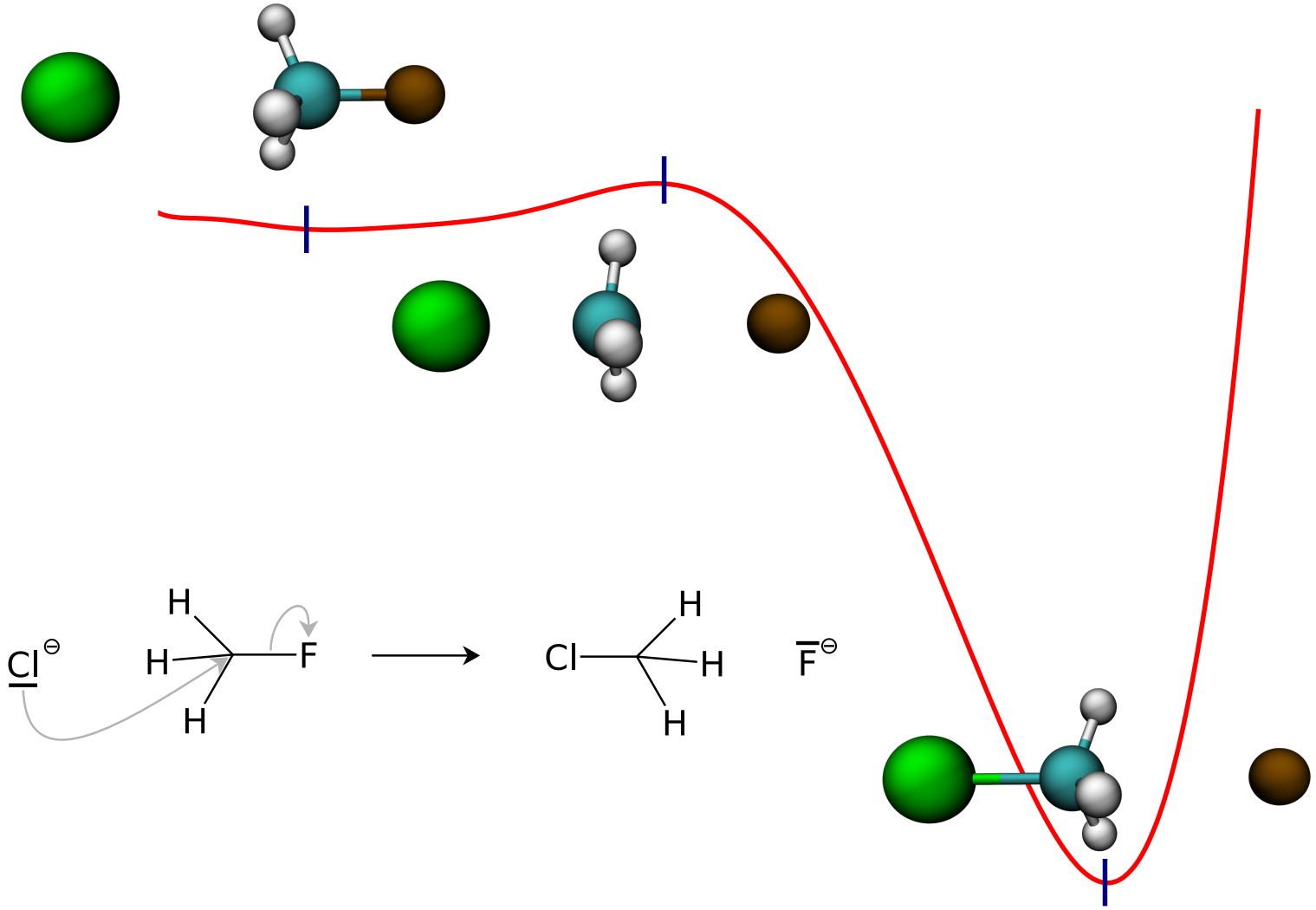
23rd May 2019

https://michael-herbst.com/talks/2019.05.23_software_development_lille.pdf

Contents

Contents

Describing chemistry



Solving the Schrödinger equation: How hard can it be?

- Main ingredient: Min-max principle

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

where $S \subset H^1(\mathbb{R}^{3N}, \mathbb{C})$ and $L^2(\mathbb{R}^{3N}, \mathbb{C})$ inner product $\langle \cdot | \cdot \rangle$

- 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?

- Work with inexact models:
  - Hartree-Fock (HF) and then Post-HF
  - Density-functional theory
- Common structure: Euler-Lagrange equations:

$$\hat{\mathcal{F}}_{\Theta^0}\psi_i^0 = \varepsilon_i \psi_i^0 \qquad \left\langle \psi_i^0 \middle| \psi_j^0 \right\rangle_1 = \delta_{ij}$$

- Need to discretise $\hat{\mathcal{F}}_{\Theta^0}$ in a basis $\{\varphi_\mu\}_\mu$:

$$F_{\mu\nu} = \left\langle \varphi_\mu \middle| \Theta^0 \varphi_\nu \right\rangle_1$$
$$C_{\mu,i} = \langle \varphi_\mu | \psi_i \rangle_1$$

- Discretised problem:

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

Which basis to choose?

- Gaussian-type orbitals
- Geminals

- Slater-type orbitals
- Sturmian-type orbitals

- ...

- Plane waves
- Augmented plane waves

- Wavelets
- Finite elements

- ...

Testing basis function types

- Obstacle: 1 Program $\simeq$ 1 basis function type

$\Rightarrow$ Basis type often burned into existing codes

$\Rightarrow$ A new program for each basis type just to try it?

- Structure of SCF problem:

$$\mathbf{FC} = \mathbf{SC}\ \mathrm{diag}(\varepsilon_1, \ldots, \varepsilon_n)$$

$\Rightarrow$ Independent of basis choice

$\Rightarrow$ It should be sufficient to swap the integral backends!

Testing basis function types

- Obstacle: 1 Program $\simeq$ 1 basis function type

$\Rightarrow$ Basis type often burned into existing codes

$\Rightarrow$ A new program for each basis type just to try it?

- Structure of SCF problem:

$$\mathbf{FC} = \mathbf{SC}\ \mathrm{diag}(\varepsilon_1, \ldots, \varepsilon_n)$$

$\Rightarrow$ Independent of basis choice

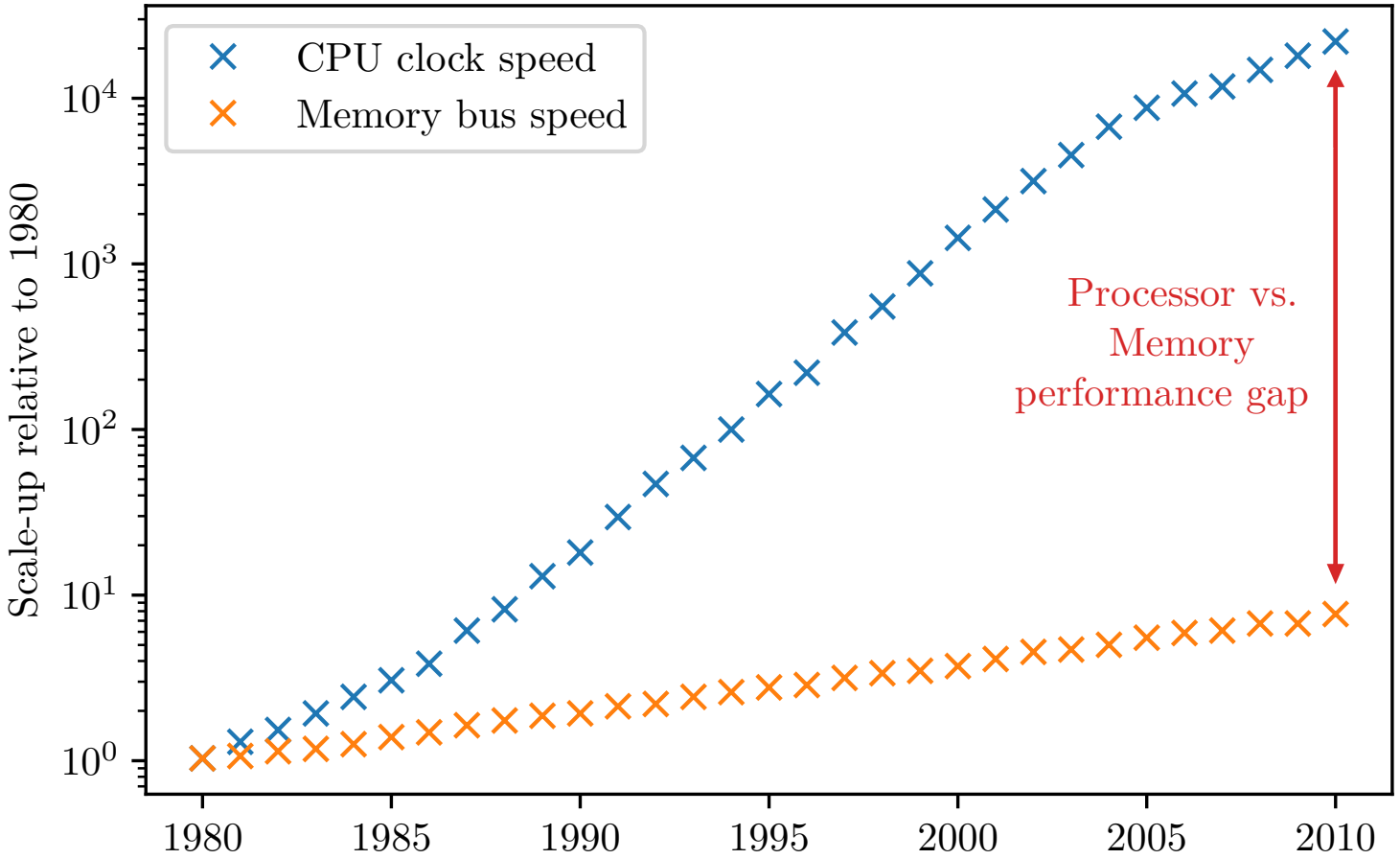
$\Rightarrow$ It should be sufficient to swap the integral backends!

Which computational back end?

- Compute clusters become increasingly Heterogeneous:
  - Multi-core and multi-CPU nodes
  - GPUs
  - FPGAs
- Strength and weaknesses differ

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

Memory versus processor developments



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

Support and popularity of programming languages?

- TIOBE index: Popularity based on searches:

3	C++	8.095%
4	python	7.830%
27	FORTRAN	0.518%
43	julia	0.218%

Source: https://www.tiobe.com/tiobe-index/, May 2019

- Popularity based on repos on github:

2	python	52933
4	C++	20537
36	julia	992
49	FORTRAN	323

Source: https://github.com/oprogramador/github-languages, Feb 2019
Counted only repos with at least 10 stars any commit since 1 Jan 2018

Lessons

- High-performance architecture constantly changes
- Impact on all levels:
  - Approximate model
  - Discretisation / basis functions
  - Algorithmic approach to numerics
  - Hardware / accelerators
  - Programming language choice

$\Rightarrow$ Need to keep trying new stuff

$\Rightarrow$ Trying should take as little time as possible

Total time cost

- Total time cost is
  - Runtime of the production calculation

Total time cost

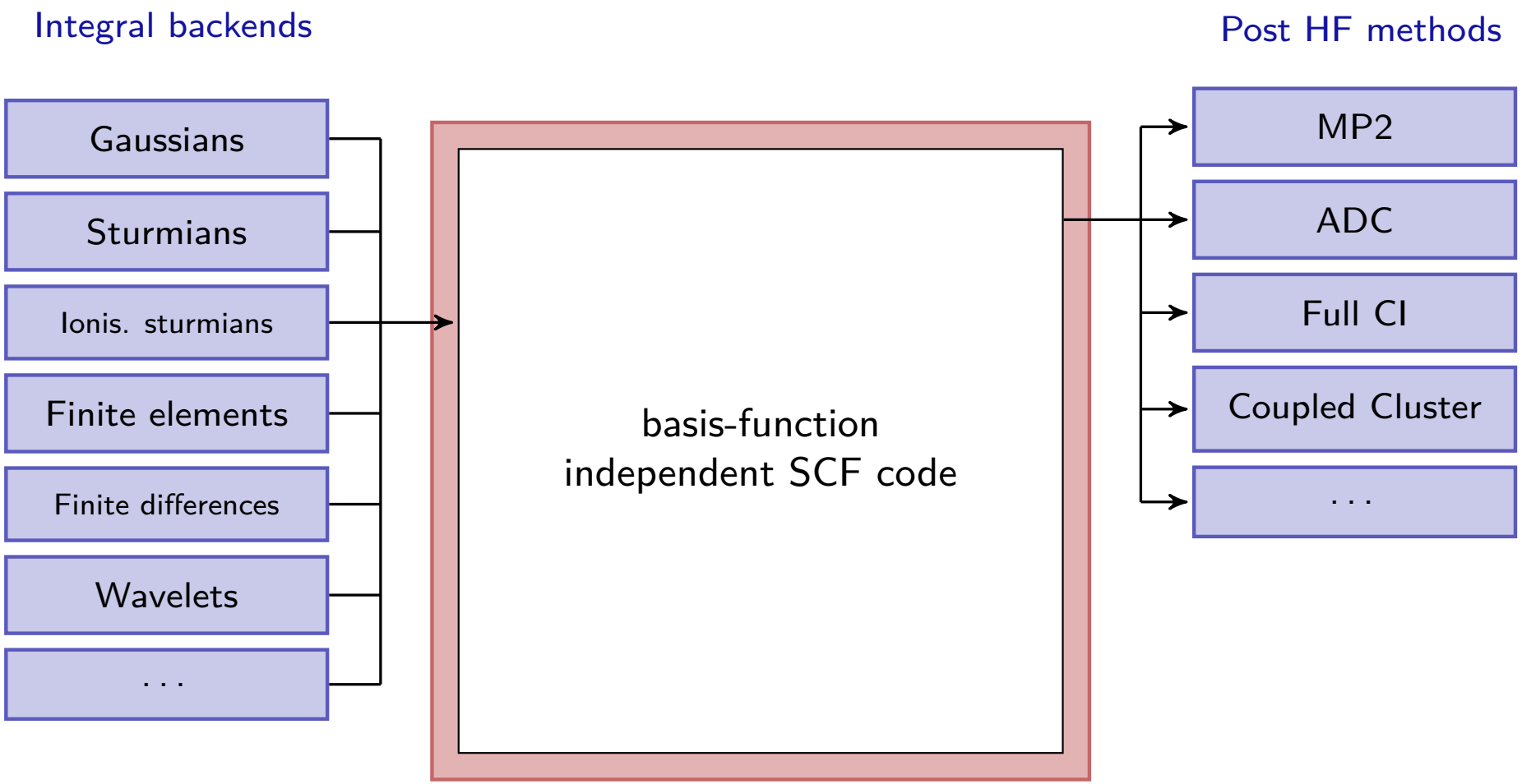
- Total time cost is sum of time for:
  - Understanding existing code
  - Implementing the new feature
  - Finding and fixing all the bugs
  - Optimising code performance
  - Runtime of the production calculation
  - Analysing and visualising results
  - Maintaining the code over its lifetime

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

Contents

Aims of molsturm



Achievements of molsturm

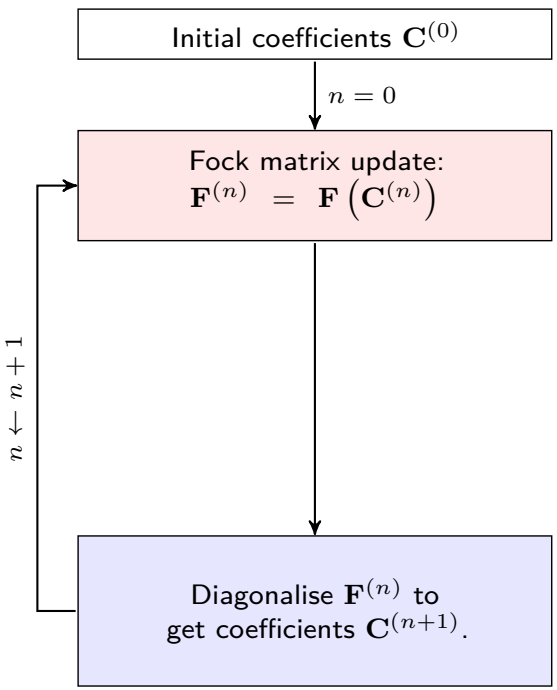
- Basis-function independent design
  - Plug and play new discretisations
  - Basis-type agnostic SCF procedure
- Easy-to-use interfaces
  - Integrate with existing code (e.g. Post-HF, python)
  - Avoid reinventing the wheel
  - Rapid prototyping, testing and analysis

⇒ Explore methods across basis function types[1,2]

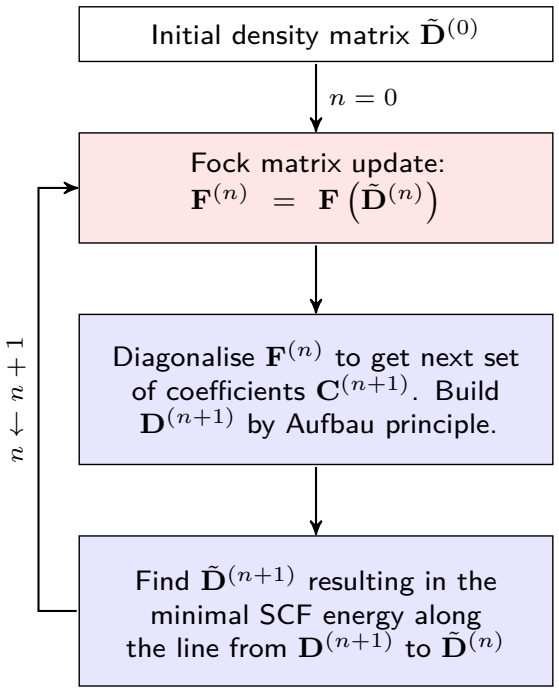
[1] M. F. Herbst, A. Dreuw and J. E. Avery. J. Chem. Phys., **149**, 84106 (2018)

[2] M. F. Herbst. Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg (2018)

Two-step structure of SCF algorithms



Roothaan scheme

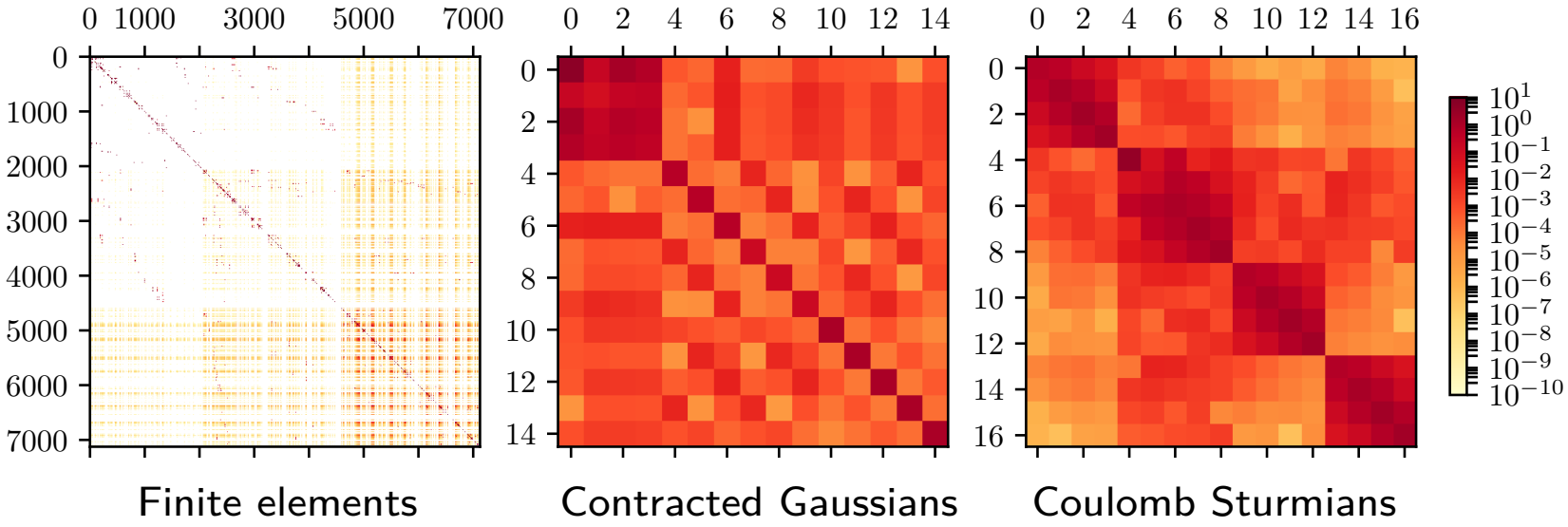


Optimal damping algorithm

- Fock update
- Coefficient update
  (density matrix update)

$\Rightarrow$ Need to be basis-type independent

Challenge: Deviating Fock matrix structures



- Required numerical procedures differ
- Details should be hidden from SCF
- Focus on HF, but our approach extends to DFT

Solution: Contraction-based methods

- Contraction-based methods
  - Avoid storing matrices
  - Employ iterative, subspace-based algorithms
  - Contraction expressions (e.g. matrix-vector products)
  - Common in Post-HF: *Working equations*

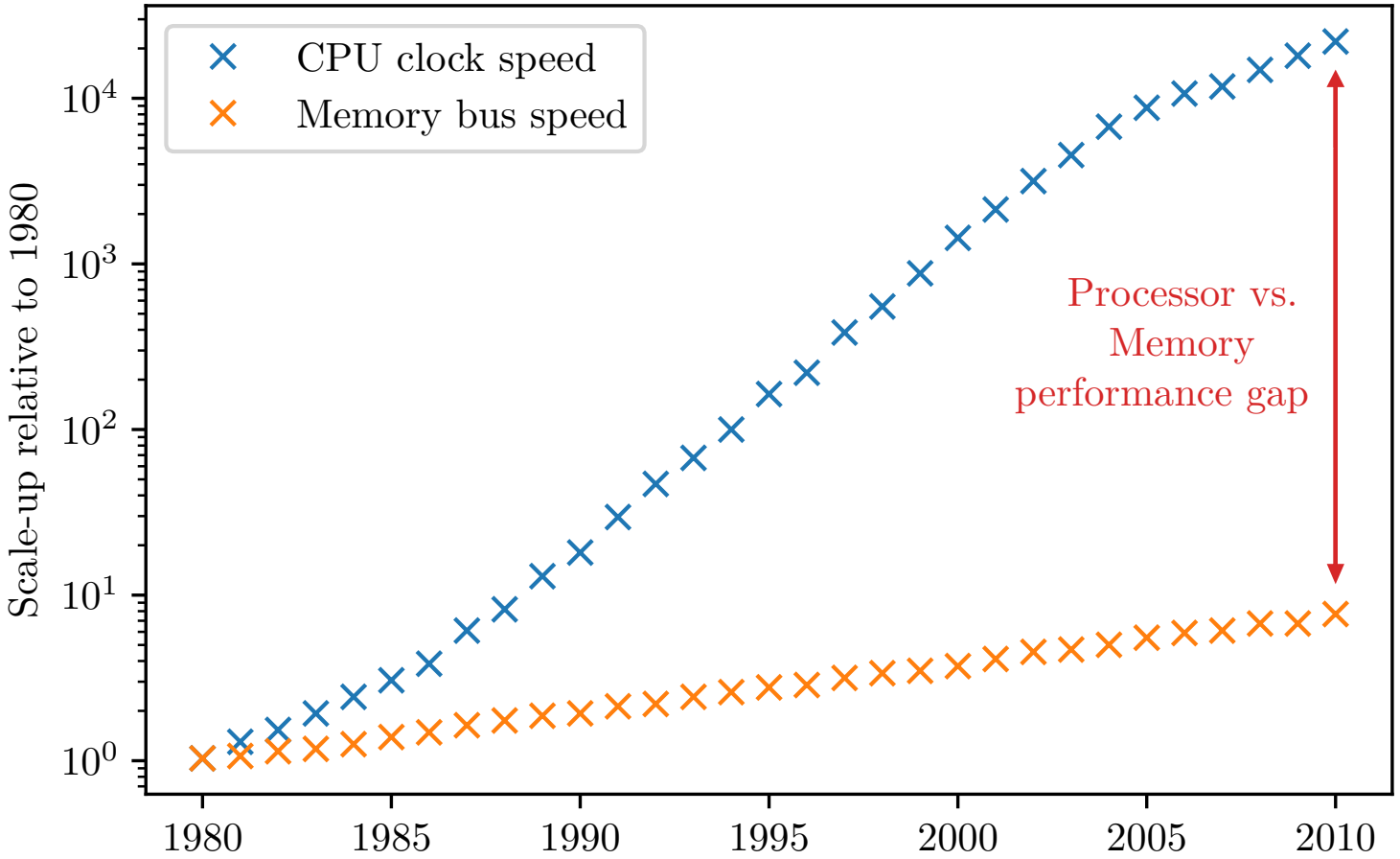
⇒ SCF code only needs Fock contraction

⇒ Hide discretisation details inside Fock object

⇒ Flexible to exploit discretisation-specific properties

⇒ Multiplex on numerical back end (`lazyten`)

Reminder: Memory versus processor developments



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

Lazy matrices

- Contraction expressions dressed as a matrix (physical intuition)
- Build and pass Fock expression tree to SCF
- Lazy evaluation:

$$\mathbf{F} = \mathbf{h} + \mathbf{J} - \mathbf{K}$$

$$\vdots$$

Iterative solver

$$\underline{\boldsymbol{y}} = \mathbf{F}\underline{\boldsymbol{x}}$$

$$\boxed{\underline{\boldsymbol{y}}} = \boxed{\mathbf{F}}\,\boxed{\underline{\boldsymbol{x}}} = \boxed{\begin{array}{c} - \\ / \quad \backslash \\ + \quad \mathbf{K} \\ / \quad \backslash \\ \mathbf{h} \quad \mathbf{J} \end{array}}\,\boxed{\underline{\boldsymbol{x}}} = (\mathbf{h} + \mathbf{J} - \mathbf{K})\,\underline{\boldsymbol{x}}$$

- Idea: Integral back end provides lazy matrix terms

Contraction-based, two-step SCF

Fock expression	Lazy matrix, sum of integral terms
Coefficient update	Iterative solvers
Fock update	Replace coefficients in expression tree

Achieve basis-function independence:

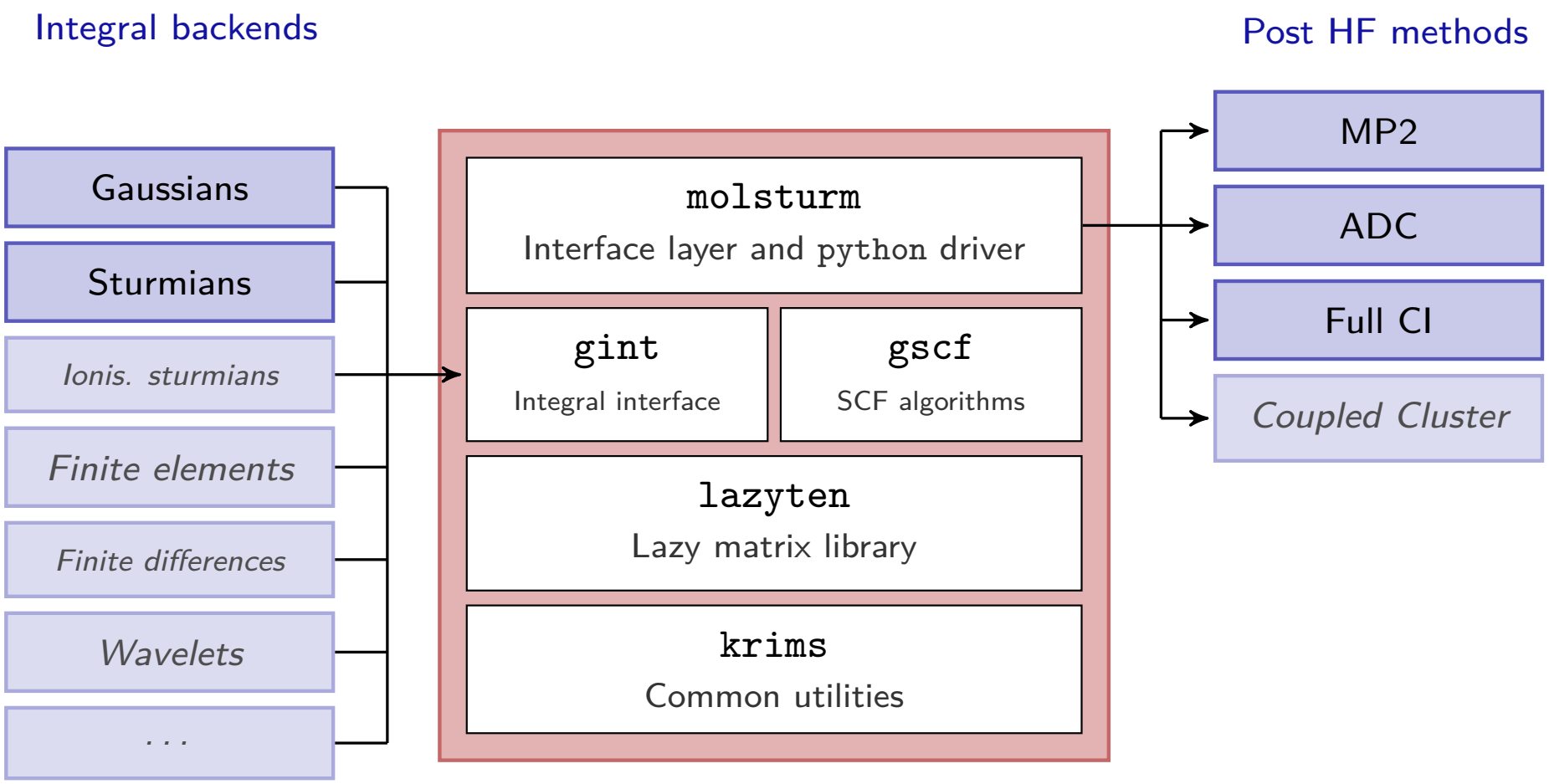
- Lazy matrices: Abstraction of integrals / SCF / numerics
- Integral back end: Controls evaluation of contractions

⇒ Decides integral data production and consumption

⇒ Transparent to SCF

⇒ May exploit discretisation-specific properties

molsturm structure



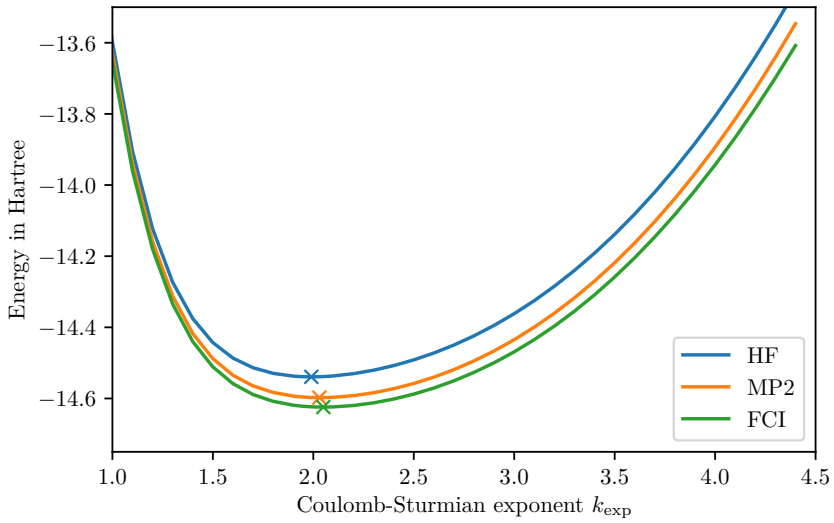
molsturm interface: CCD residual (parts)

$$
\begin{aligned}
r_{ij}^{ab} = &-\frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{mn}^{af}\, t_{ij}^{eb} + \frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{mn}^{bf}\, t_{ij}^{ea} - \frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{in}^{ef}\, t_{mj}^{ab} \\
&+\frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{jn}^{ef}\, t_{mi}^{ab} + \frac{1}{4}\sum_{mnef}\langle mn||ef\rangle\, t_{mn}^{ab}\, t_{ij}^{ef} + \frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{im}^{ae}\, t_{jn}^{bf} \\
&-\frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{jm}^{ae}\, t_{in}^{bf} - \frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{im}^{be}\, t_{jn}^{af} + \frac{1}{2}\sum_{mnef}\langle mn||ef\rangle\, t_{jm}^{be}\, t_{in}^{af}
\end{aligned}
$$

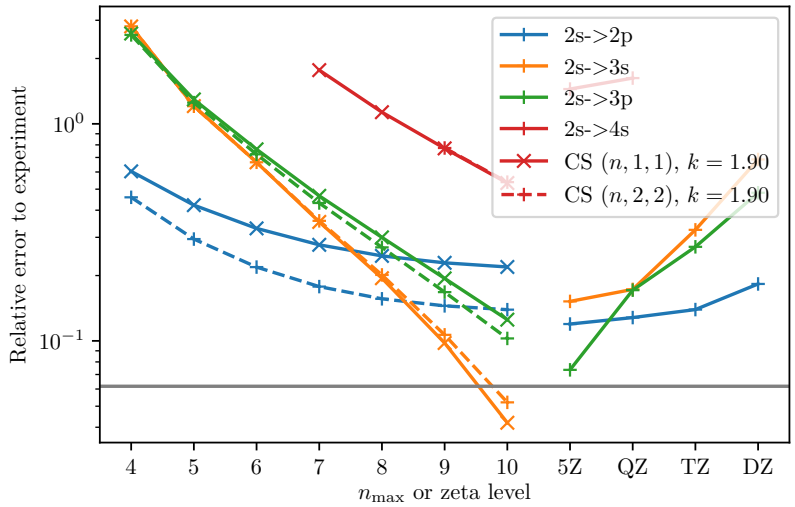
```
eri_phys = state.eri.transpose((0, 2, 1, 3))
eri = eri_phys - eri_phys.transpose((1, 0, 2, 3))
res = \
    - 0.5  * einsum("mnef,manf,iejb->iajb", eri.block("oovv"), t2, t2) \
    + 0.5  * einsum("mnef,mbnf,ieja->iajb", eri.block("oovv"), t2, t2) \
    - 0.5  * einsum("mnef,ienf,majb->iajb", eri.block("oovv"), t2, t2) \
    + 0.5  * einsum("mnef,jenf,maib->iajb", eri.block("oovv"), t2, t2) \
    + 0.25 * einsum("mnef,manb,iejf->iajb", eri.block("oovv"), t2, t2) \
    + 0.5  * einsum("mnef,iame,jbnf->iajb", eri.block("oovv"), t2, t2) \
    - 0.5  * einsum("mnef,jame,ibnf->iajb", eri.block("oovv"), t2, t2) \
    - 0.5  * einsum("mnef,ibme,janf->iajb", eri.block("oovv"), t2, t2) \
    + 0.5  * einsum("mnef,jbme,ianf->iajb", eri.block("oovv"), t2, t2)
```

molsturm interface: Linked codes

Coulomb-Sturmian based MP2 and FCI



Coulomb-Sturmian and Gaussian based ADC(2)



- FCI from `pyscf`[1]
- Coulomb Sturmians from `sturmint`[2]

- ADC(2) from `adcc`[3]
- Gaussians from `libint`[4] or `libcint`[5]

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

[2] J. E. Avery and M. F. Herbst. `https://molsturm.org/sturmint` (2018)

[3] M. F. Herbst et al. *adcc: Seamlessly connect your host application to ADC*. In preparation.

[4] E. Valeyev et al. *evaleev/libint: 2.3.1* (2017).

[5] Q. Sun. J. Comput. Chem., **36**, 1664 (2015)

molsturm demo

DEMO

of a gradient-free
geometry optimisation

molsturm Structure

- python Code with C++ core
- Contraction-based SCF using lazy matrices
- Abstraction layer to linear algebra and basis functions
- Unit tests
- Not fast, but general
- Integration with python ecosystem for:
  - Additional features (e.g. CCD, geometry optimisation)
  - Data analysis (e.g. pandas)
  - Plotting (e.g. matplotlib)
  - ...

Contents

DFTK.jl overview

- Toolkit for periodic electronic structure problems
- Focus on plane-wave discretisations:

$$\forall \underline{\boldsymbol{k}} : \ \varphi_{\underline{\boldsymbol{G}}} = e^{\imath \underline{\boldsymbol{k}} \cdot \underline{\boldsymbol{x}}} \, e^{\imath \underline{\boldsymbol{G}} \cdot \underline{\boldsymbol{x}}}$$

- Density-functional theory
- Toolkit:
  - Minimalistic code base (not a program package)
  - Use existing libraries and codes
  - Facilitate integration elsewhere
- Written in `julia`

DFTK.jl aims and examples of applicability

- Accessible to mathematicians, physicists, computer scientists

- Use DFTK.jl primitives to build new models
- Numerical analysis of algorithms
  - Convergence behaviour and proof
  - Existence / uniqueness / optimality of solutions

  ⇒ Often need numerical experiments
- Experiment with DFT codes in modern HPC environments

⇒ Support both toy problems and full-scale applications

Why julia?

- Very recent: v1.0. released August 2018
- Compiled scripting language
- JIT with LLVM back end
- High-level syntax and dynamical
- Strongly typed
- Rich interoperability: FORTRAN, C, C++, python, R, ...

≈ python with deeply integrated numpy

Why julia? (2)

- Key concept: Multiple dispatch
- First call: Function compiled *exactly* for argument types

⇒ First call is slow (JIT compilation)

⇒ 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 . . .

julia DEMO

DEMO

A 5-min introduction to julia

A word about 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
C	clang	8.0
C	clang -O3	1.1
C	clang -O3 -march=native	0.8
julia	array operations	3.3
julia	loops	1.9
julia	loops, no bounds check	0.5

- Best out of five on my laptop
- Heat equation example (courtesy Antoine Levitt)
- Used software: gcc 8.3, clang 7.0.1, python 3.7, numpy 1.16.2, julia 1.0.3

julia summary

- More functional, less OOP
- Modern HPC in high-level syntax:
  - Threading, vectorisation, distributed memory parallelism
- Great features in the pipeline:
  - Adjoint-mode automatic differentiation
  - GPU back ends
  - Machine learning
  - Large-scale data analysis
- Still able to use all code from `python`, `C`, `C++`, R, ...

Status of DFTK.jl

- Development start: 01/01/2019
- First working plane-wave LDA: 31/03/2019
  - Interface to libxc
  - DIIS-based SCF based on NLsolve.jl and IterativeSolvers.jl
  - Ground-state plane-wave calculations in 3D
  - Laptop-level parallelism
  - Analytic potentials and LDA-DFT
- Currently: Redesign in progress
  - https://github.com/mfherbst/DFTK.jl

Outlook

- Flexibility in the problem dimensions
- Mixed and elevated precision
- Forces and stresses
- Response and properties
- Mixed basis or grid methods
- Mathematical analysis of SCF convergence
- Error estimates

adcc: python-driven ADC for SCF codes

- Joint project with M. Scheurer, T. Fransson, D. R. Rehn, A. L. Dempwolff (Dreuw group, Heidelberg University)
- Algebraic-diagrammatic construction (ADC) approach to electronic excitations
- python layer:
  - Connection to SCF drivers (4 codes so far)
  - Numerical algorithms (eigensolver / linear response)
  - Controls computational workflow
- libtensor[1] C++ library: Heavy tensor-contractions
- Part of Gator framework for computational spectroscopy[2]

[1] E. Epifanovsky, M. Wormit, T. Kuś et al. J. Comput. Chem., **34**, 2293 (2013)

[2] D. R. Rehn, Z. Rinkevicius, M. F. Herbst, et. al. *Gator: A python-driven wave-function correlated program for spectroscopy calculations.* In preparation.

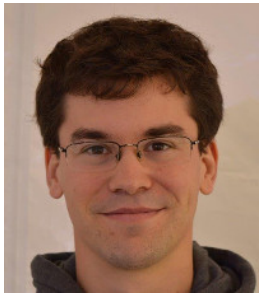
Summary: molsturm and DFTK.jl

- Modern HPC architectures are heterogeneous:
  - No single, best choice of algorithms, discretisations, methods
  - ⇒ Need rapid prototyping approaches
- molsturm: Molecular problems
  - Modular and light-weight structure, python interface
  - Contraction-based, basis-function independent SCF
  - ⇒ Plug-and-play basis-function types or Post-HF methods
- DFTK.jl: Periodic problems
  - Tool kit on top of julia ecosystem
  - Aid for numerical analysis and proof
  - julia allows to adapt to HPC environments
  - ⇒ Both toy problems and full-scale applications

Acknowledgements

James Avery

Andreas Dreuw



Antoine Levitt

Eric Cancès

Inria

École des Ponts ParisTech

SORBONNE UNIVERSITÉ

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

IWR Interdisciplinary Center for Scientific Computing

HGS MathComp

Questions?

Thesis: https://michael-herbst.com/phd-thesis.html

molsturm: M. F. Herbst, A. Dreuw and J. E. Avery. J. Chem. Phys., **149**, 84106 (2018)

https://molsturm.org

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

Email: michael.herbst@enpc.fr

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

Contraction-based methods: Overview

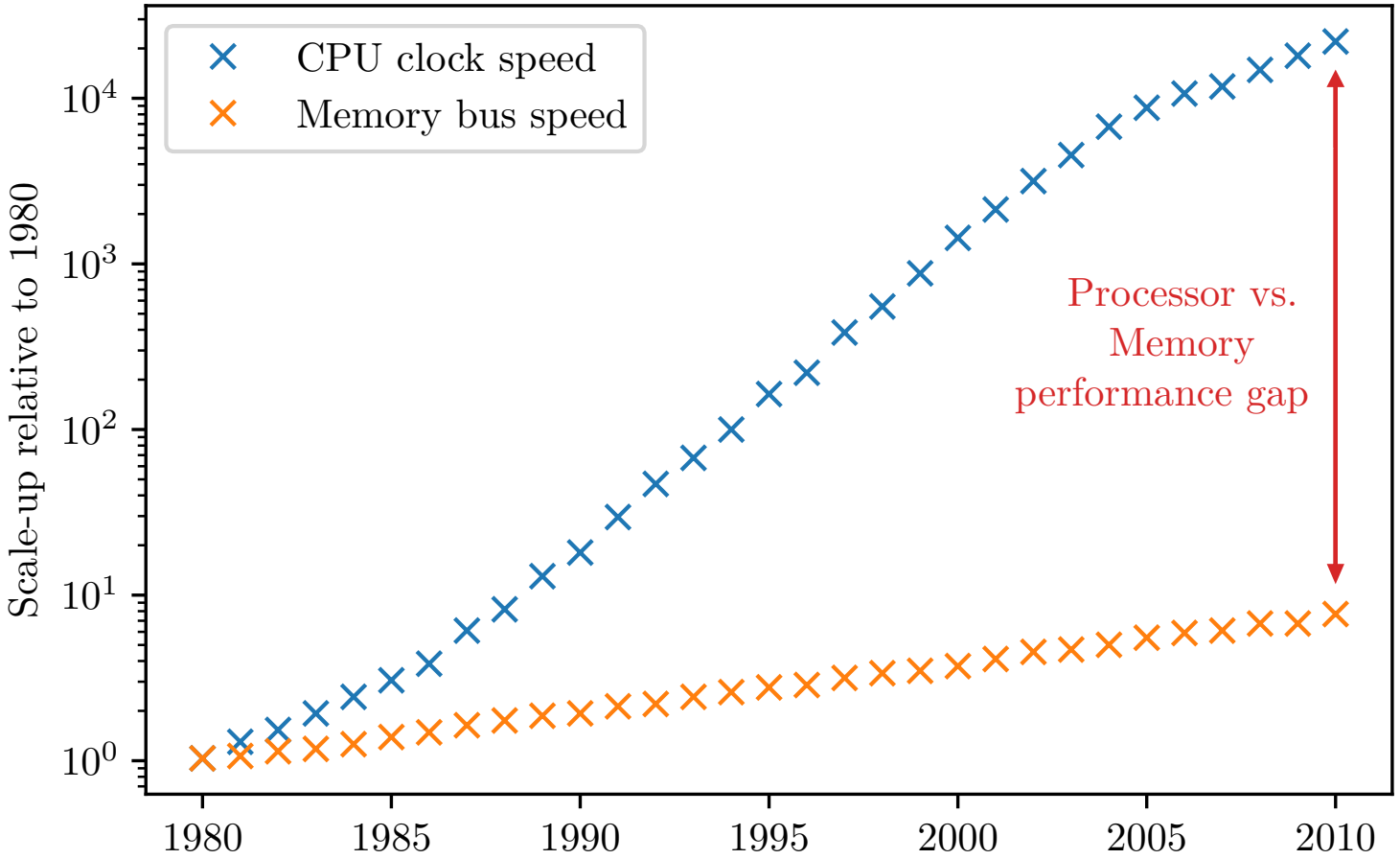
Advantages

- Maximum flexibility at the point of evaluation
- Parallelisation easier
  - ⇒ Less data management
  - ⇒ Easier modularisation
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than contraction-functions
- More computations
  - ⇒ Need efficient contraction schemes for the contraction
  - ⇒ Algorithms more complex

Contraction-based methods: Overview



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

Contraction-based methods: Overview

Storage layer	Latency /ns	FLOPs
L1 cache	0.5	13
L2 cache	7	180
Main memory	100	2600
SSD read	$1.5 \cdot 10^4$	$4 \cdot 10^5$
HDD read	$1 \cdot 10^7$	$3 \cdot 10^8$

Data from
`https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html`
FLOPs for a Sandy Bridge 3.2GHz CPU with perfect pipelining

Contraction-based methods: Overview

Advantages

- Maximum flexibility at the point of evaluation
- Parallelisation easier
  - ⇒ Less data management
  - ⇒ Easier modularisation
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than contraction-functions
- More computations
  - ⇒ Need efficient contraction schemes for the contraction
  - ⇒ Algorithms more complex

Contraction-based methods: Flexibility

- Compare

1. $$K_{\kappa\lambda} = \sum_{\mu\nu,i} \langle \kappa\nu || \mu\lambda \rangle \, C_{\mu,i} C_{\nu,i}$$
2. $$y_\kappa = \sum_\lambda K_{\kappa\lambda} x_\lambda$$

with directly

$$y_\kappa = \sum_{\lambda\mu\nu,i} \langle \kappa\nu || \mu\lambda \rangle \, C_{\mu,i} C_{\nu,i} x_\lambda$$

- Reordering terms
- Exploit known symmetries in x_λ, $\langle \kappa\nu || \mu\lambda \rangle$
- Exploit index selection rules
- $\mathbf{K}$ like a matrix with state $\mathbf{C}$

Lazy matrix evaluation

- Actual expression in source code

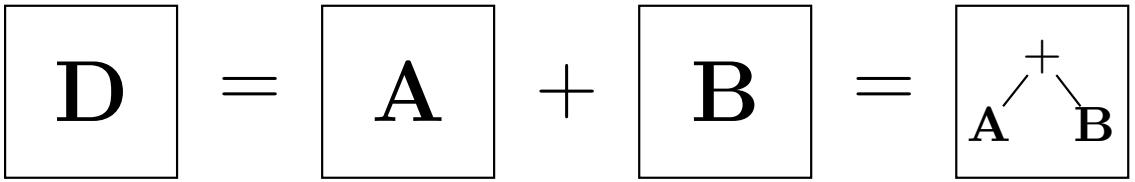
$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

Lazy matrix evaluation

- Actual expression in source code

$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

- Performed operation



Lazy matrix evaluation

- Actual expression in source code

$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

- Performed operation

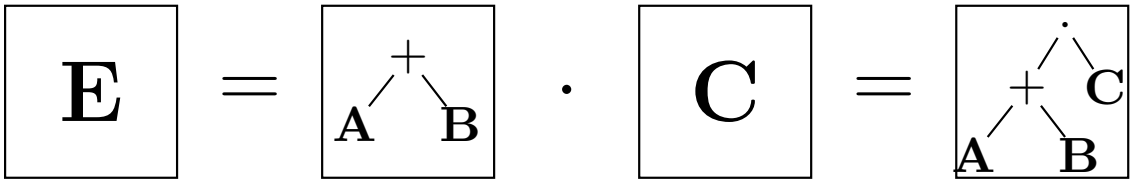
$$\boxed{\mathbf{E}} = \boxed{\mathbf{D}} \cdot \boxed{\mathbf{C}}$$

Lazy matrix evaluation

- Actual expression in source code

$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

- Performed operation

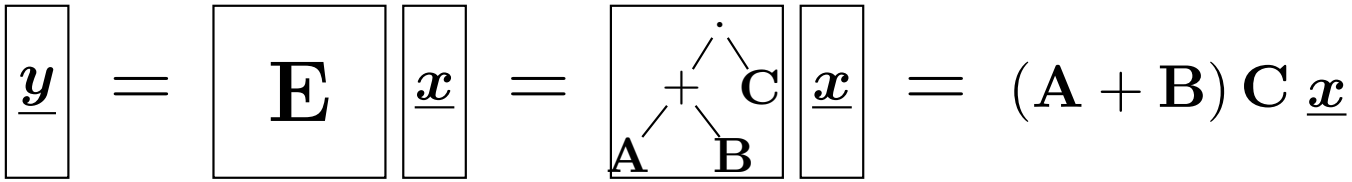


Lazy matrix evaluation

- Actual expression in source code

$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

- Performed operation



lazyten: Lazy matrix library