Lazy matrices for contraction-based algorithms

Michael F. Herbst
michael.herbst@iwr.uni-heidelberg.de
https://michael-herbst.com

Interdisziplinäres Zentrum für wissenschaftliches Rechnen
Ruprecht-Karls-Universität Heidelberg

4th October 2017

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

IWR
Interdisciplinary Center for Scientific Computing

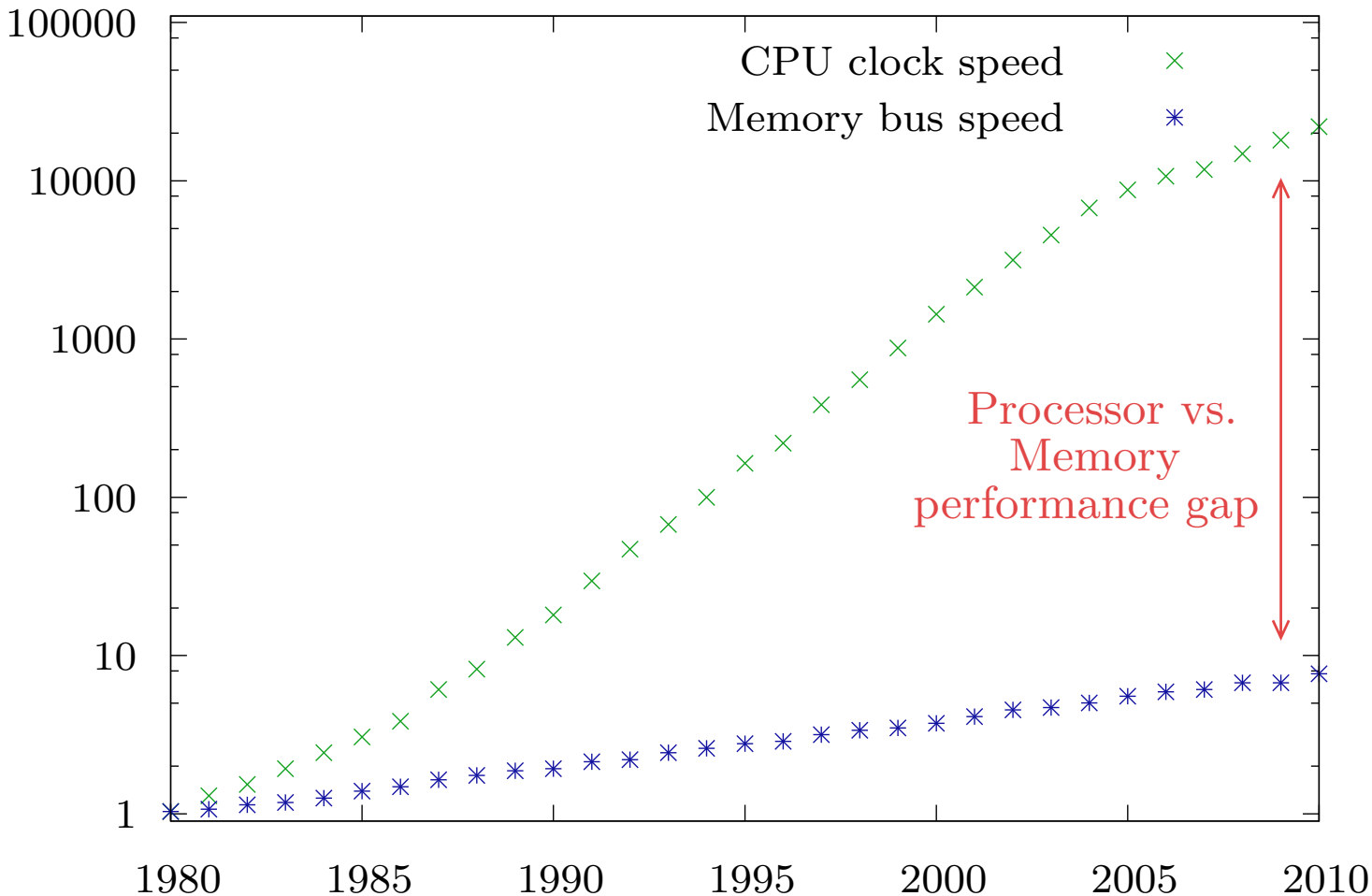
Contents

Contents

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

IWR Interdisciplinary Center for Scientific Computing

Processor vs. memory performance improvement



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

Hartree-Fock equations

- Hartree-Fock equations

$$\left(-\frac{1}{2}\Delta + \hat{\mathcal{V}}_{\text{Nuc}} + \hat{\mathcal{V}}_{\text{H}}\left[\{\psi_f\}_{f\in I}\right] + \hat{\mathcal{V}}_{\text{x}}\left[\{\psi_f\}_{f\in I}\right] - \varepsilon_f\right)\psi_f(\underline{\boldsymbol{r}}) = 0$$

with

$-\frac{1}{2}\Delta$	Kinetic energy of electrons
$\hat{\mathcal{V}}_{\text{Nuc}}$	Electron-nuclear interaction
$\hat{\mathcal{V}}_{\text{H}}\left[\{\psi_f\}_{f\in I}\right]$	Hartree potential
$\hat{\mathcal{V}}_{\text{x}}\left[\{\psi_f\}_{f\in I}\right]$	Exchange potential

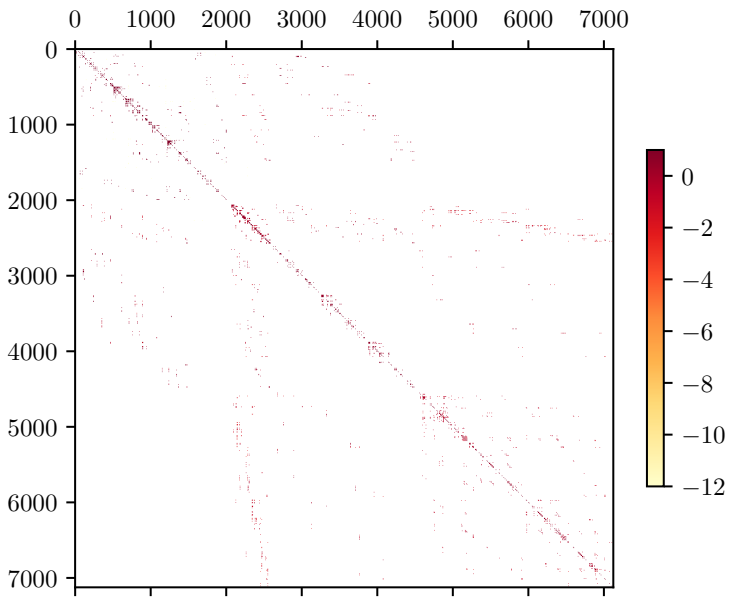
- Non-linear system of partial differential equations
- Non-linear eigenproblem for eigenpairs $\{(\varepsilon_f, \psi_f)\}_{f\in I}$

Finite-element discretisation

- Finite elements: Piecewise polynomials with support only on a few neighbouring *cells*

$\Rightarrow$ Need many finite elements ($> 10^6$)

- Typically sparse matrix structures:

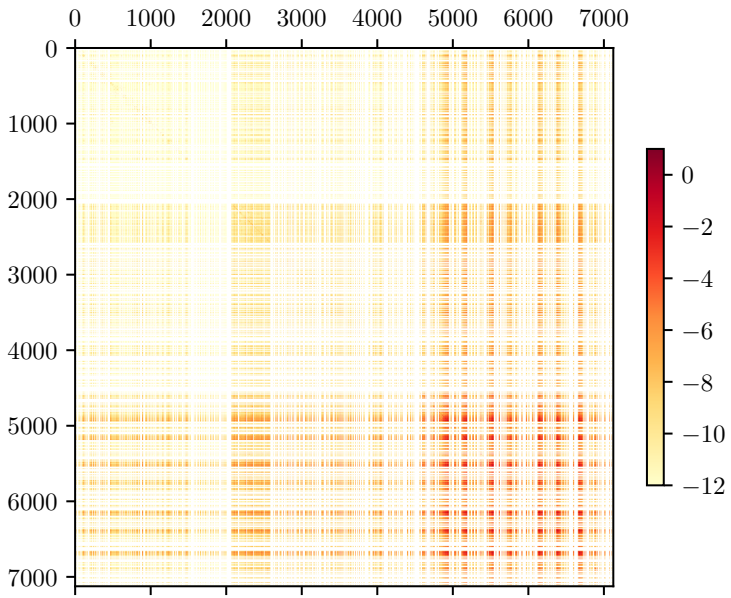


Typical discretisation of $-\frac{1}{2}\Delta + \hat{\mathcal{V}}_{\mathrm{Nuc}} + \hat{\mathcal{V}}_{\mathrm{H}}$

Finite-element discretisation

Caveat

- But … $\hat{\mathcal{V}}_{\mathrm{x}}$ is so-called *non-local*:



K: Discretisation of $\hat{\mathcal{V}}_{\mathrm{x}}$

- Building **K** takes $\Omega(N^2)$ time and $\mathcal{O}(N^2)$ storage
- Typically $10^6 \cdot 10^6$ elements $\approx 8\,\mathrm{TB}$ storage

Finite-element discretisation

contraction-based scheme

- Iterative solvers only need matrix-vector products
- Matrix-vector product of $\mathbf{K}$: Theoretically $\mathcal{O}(N)$

$\Rightarrow$ contraction-based or matrix-free algorithm:

- Never build $\mathbf{K}$ in storage
- Use expression for $\mathbf{K}$ to directly form contraction of matrix to vectors

contraction-based SCF

Possible problems

- For SCF we will need

$$\mathbf{F} = \mathbf{T} + \mathbf{V}_{\mathrm{Nuc}} + \mathbf{J} + \mathbf{K} + \cdots$$

- Requirements differ:
  - Optimal storage scheme
  - Optimal contraction scheme
  - Approximations / costs

$\Rightarrow$ `contraction` expression complicated to code

$\Rightarrow$ We would like to stay flexible

Characteristics of `contraction`-based algorithms

Advantages

- Scaling (storage and time) reduced — in examples to $\mathcal{O}(N)$
- Parallelisation easier
  - $\Rightarrow$ Less data management
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than `contraction`-functions
- More computations
  - $\Rightarrow$ Need efficient contraction schemes for the `contraction`
  - $\Rightarrow$ Algorithms more complicated

Characteristics of `contraction`-based algorithms

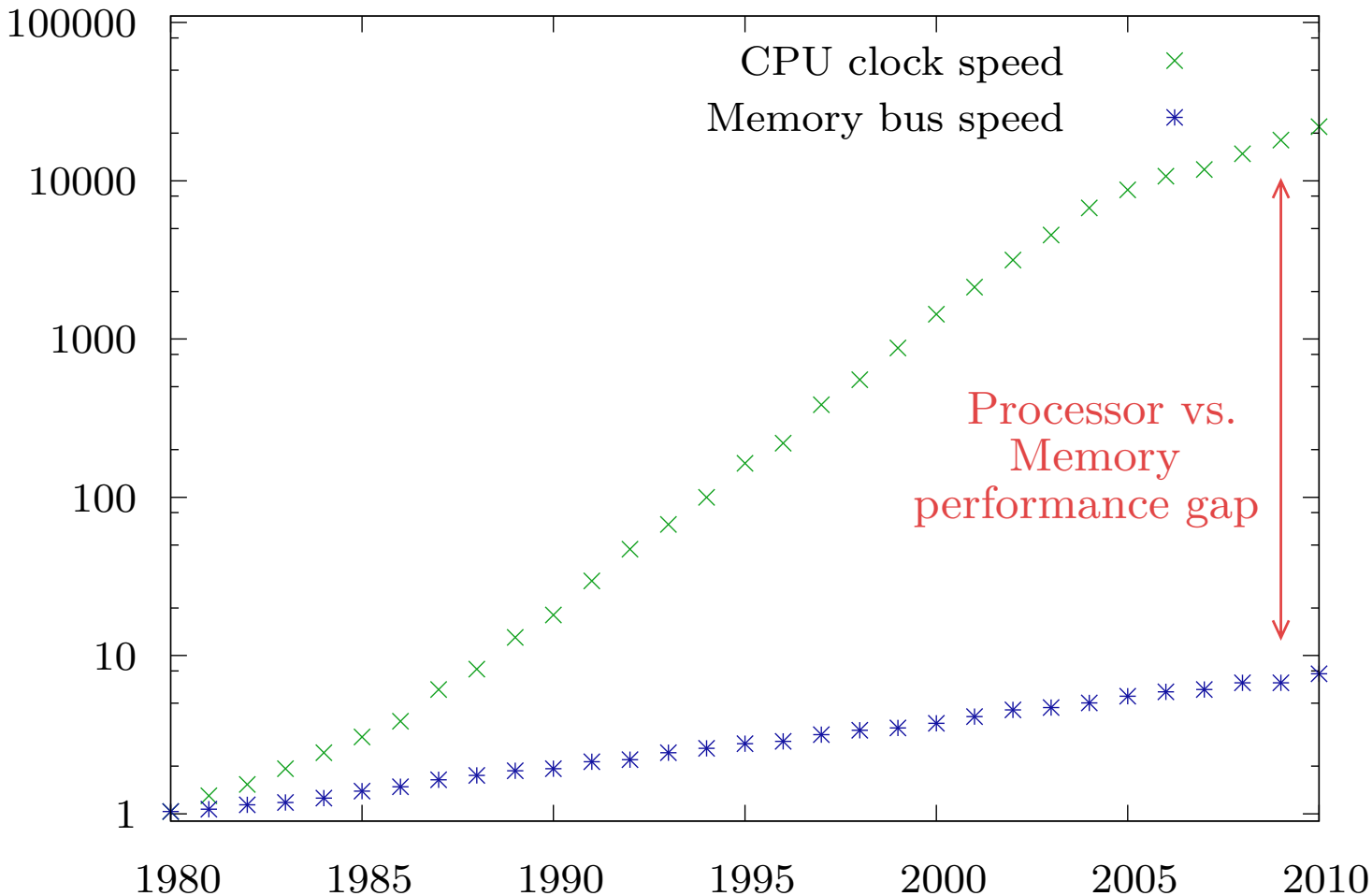
Advantages

- Scaling (storage and time) reduced — in examples to $\mathcal{O}(N)$
- Parallelisation easier
  - $\Rightarrow$ Less data management
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than `contraction`-functions
- More computations
  - $\Rightarrow$ Need efficient contraction schemes for the `contraction`
  - $\Rightarrow$ Algorithms more complicated

Characteristics of contraction-based algorithms



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

Characteristics of `contraction`-based algorithms

Advantages

- Scaling (storage and time) reduced — in examples to $\mathcal{O}(N)$
- Parallelisation easier
  - $\Rightarrow$ Less data management
- Hardware trends are in favour

Disadvantages

- Matrices more intuitive than `contraction`-functions
- More computations
  - $\Rightarrow$ Need efficient contraction schemes for the `contraction`
  - $\Rightarrow$ Algorithms more complicated

Contents

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

IWR Interdisciplinary Center for Scientific Computing

Lazy matrices

- Stored matrix: All elements reside in memory
- Lazy matrix:
  - Generalisation of matrices
    - State
    - Non-linear
    - Elements may be expressions

  $\Rightarrow$ Obtaining elements expensive

  - Evaluation of internal expression: Delayed until `contraction`
  - For convenience: Offer matrix-like interface

Using lazy matrices

- Program as usual

$$\mathbf{D} = \mathbf{A} + \mathbf{B}$$

- Build expression tree internally

$$\boxed{\mathbf{D}} = \boxed{\mathbf{A}} + \boxed{\mathbf{B}}$$

- On application:

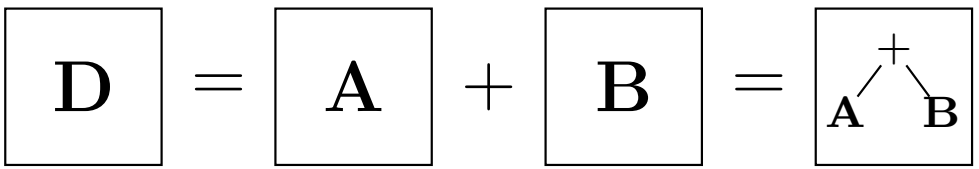
$$\mathbf{D}\underline{x} = (\mathbf{A}\underline{x}) + (\mathbf{B}\underline{x})$$

Using lazy matrices

- Program as usual

$$\mathbf{D} = \mathbf{A} + \mathbf{B}$$

- Build expression tree internally



- On application:

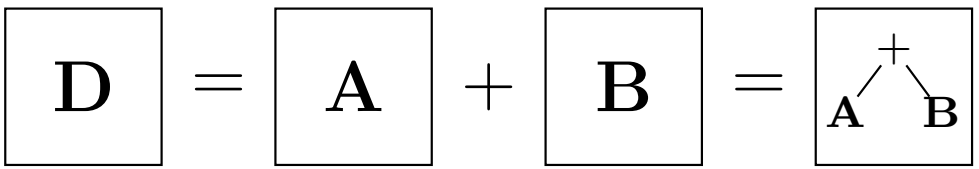
$$\mathbf{D}\underline{x} = (\mathbf{A}\underline{x}) + (\mathbf{B}\underline{x})$$

Using lazy matrices

- Program as usual

$$\mathbf{D} = \mathbf{A} + \mathbf{B}$$

- Build expression tree internally



- On application:

$$\mathbf{D}\underline{\boldsymbol{x}} = (\mathbf{A}\underline{\boldsymbol{x}}) + (\mathbf{B}\underline{\boldsymbol{x}})$$

Notes and observations

- `lazyten`: Bookkeeping for `contraction`-functions
- Programmer still sees matrices

$\Rightarrow$ Language for writing `contraction`-based algorithms

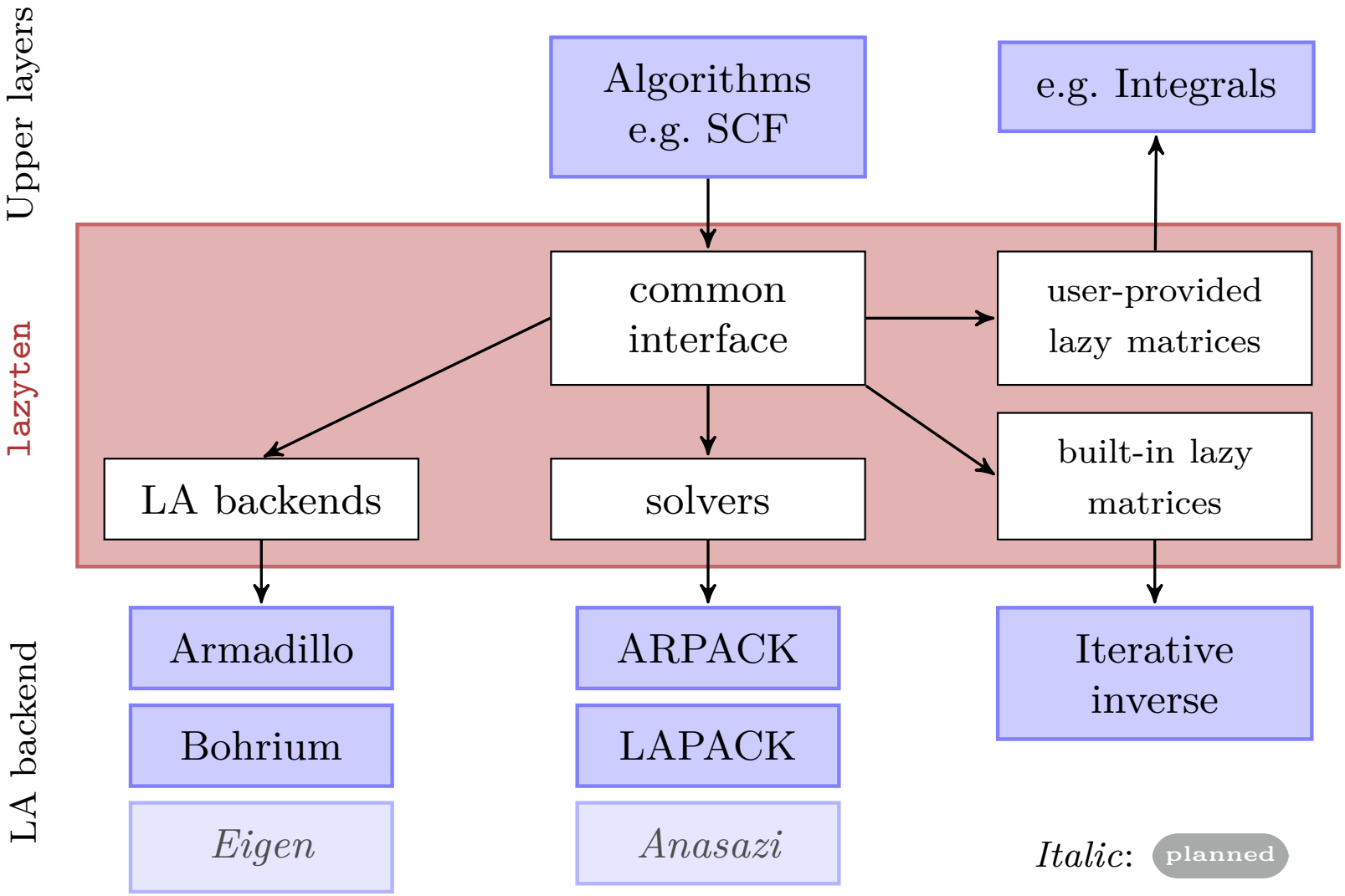
- Lazy matrices allow layered responsibility for computation, e.g. $(\mathbf{A} + \mathbf{B})\underline{\boldsymbol{x}}$
  - $\mathbf{A}\underline{\boldsymbol{x}}$ and $\mathbf{B}\underline{\boldsymbol{x}}$ decided by implementation of $\mathbf{A}$ and $\mathbf{B}$
  - $(\mathbf{A}\underline{\boldsymbol{x}}) + (\mathbf{B}\underline{\boldsymbol{x}})$ done in linear algebra backend

$\Rightarrow$ Proper modularisation between

- Higher-level algorithms
- Lazy matrix implementations
- LA backends

lazyten: Lazy matrix library

Structure of the library



lazyten in practice: Example code

- lazyten[1]: Prototype C++ implementation

```
1  typedef SmallVector<double> vector_type;
2  typedef SmallMatrix<double> matrix_type;
3  auto v = random<vector_type>(100);
4  DiagonalMatrix<matrix_type> diag(v);
5  auto a = random<matrix_type>(100,100);
6  auto b = random<matrix_type>(100,100);
7
8  // No computation: Just build expression tree
9  auto sum = diag + a;
10 auto projector = sum * inverse(sum);
11 auto tree = b - projector * b;
12
13 // Evaluate tree on application:
14 SmallVector<double> res = tree * v;
```

[1] https://lazyten.org

Contents

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

IWR Interdisciplinary Center for Scientific Computing

Lazy matrices for quantum chemistry codes

- `contraction`-based algorithms

⇒ Lower memory footprint, scaling improvements

- Readable code

⇒ Great for teaching or to play around

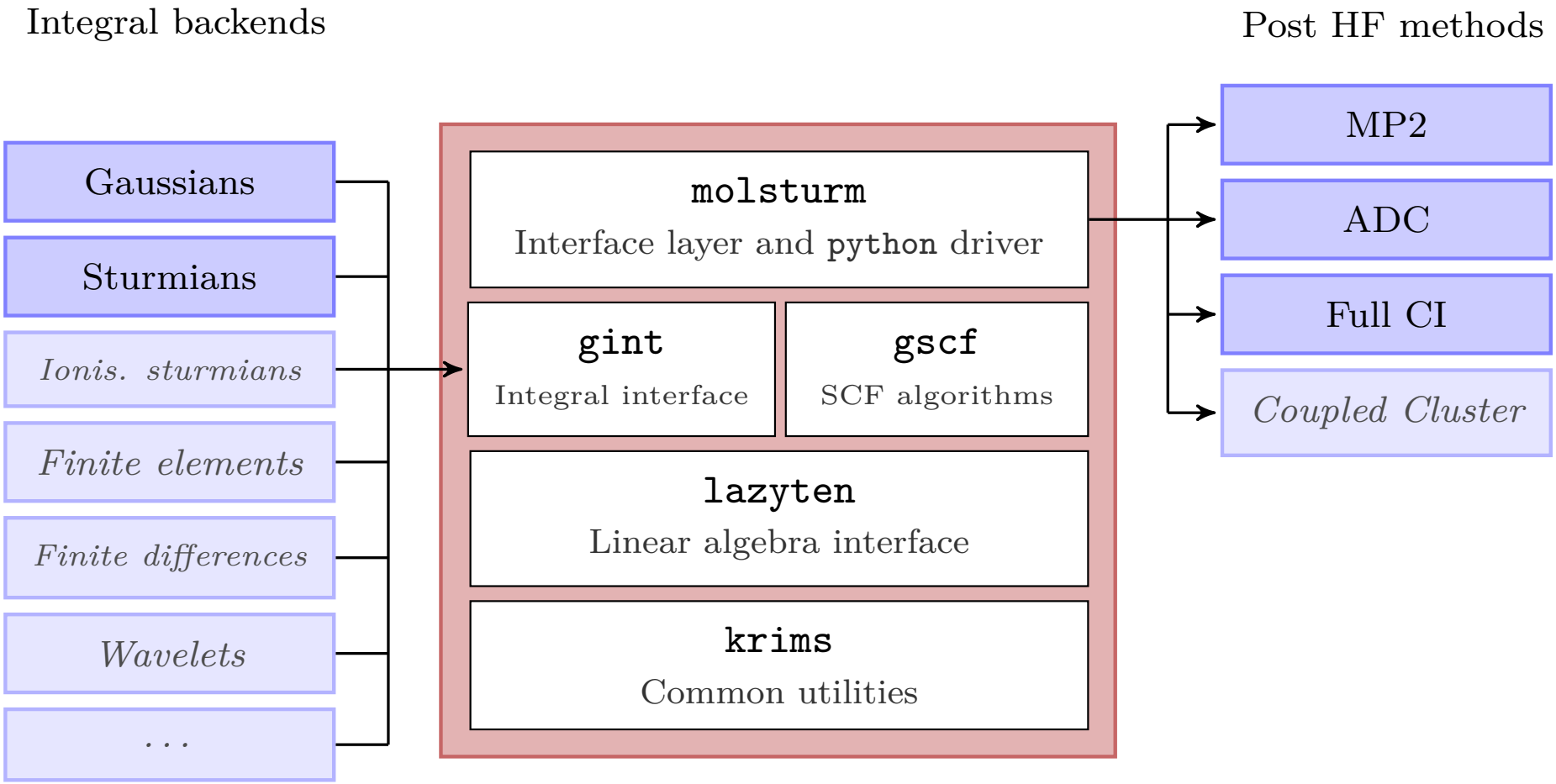
- Abstraction between integrals and SCF algorithms

⇒ Plug and play integral libraries

⇒ Swap LA backends

⇒ Basis-type independent SCF / quantum chemistry

molsturm structure



- *Italic*: planned

molsturm design

- Enables `contraction`-based SCF routines
- Flexiblity as primary goal
- Behaviour controlled via `python`
  - Keywords to change basis type or solver
  - All computed data available in `numpy` format
  - No input file, just a `python` script
- `python` utilities
  - Import / export results
  - Post-HF calculations

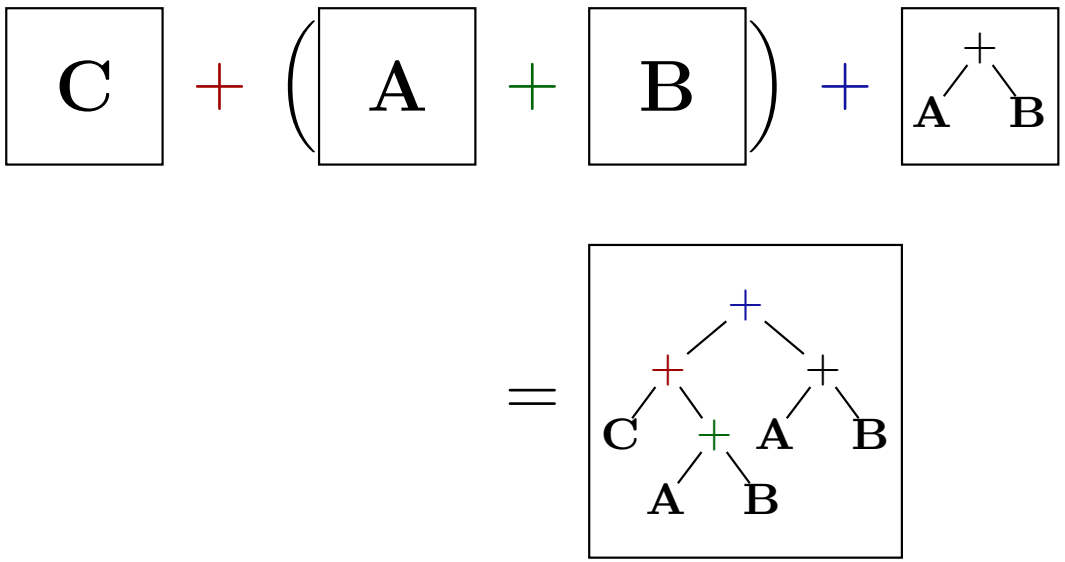
molsturm interface: CCD residual (parts)

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

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

Contents

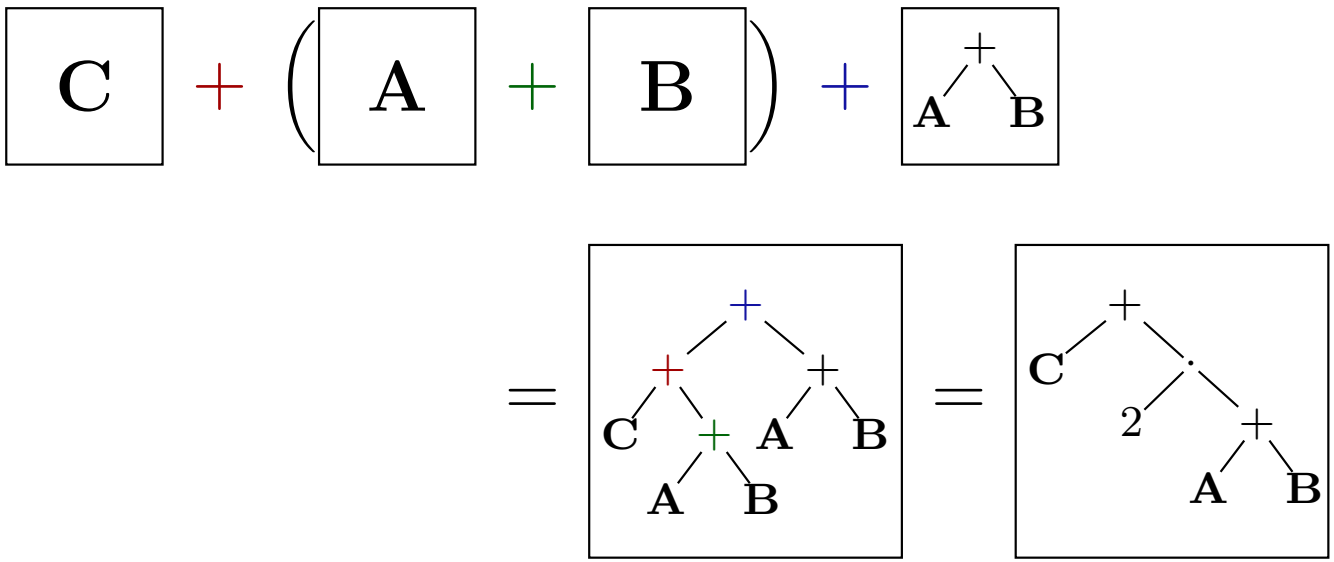
Lazy matrix expression optimisation



- Matrix expression tree ≡ abstract syntax tree ≡ DAG

⇒ May be optimised by standard methods

Lazy matrix expression optimisation



- Matrix expression tree ≡ abstract syntax tree ≡ DAG

⇒ May be optimised by standard methods

Selection of LA backend

- Right now: LA backend is compiled in
- e.g. Bohrium backend
  - Uses just-in-time (JIT) compilation
  - Very specific for hardware
  - Compilation takes time
- Better: Dynamic selection
  - Load on the expression tree
  - Availablility of backends
  - Hardware specs

Extension to lazy tensors

- `contraction` is a special tensor contraction

$\Rightarrow$ Lazy tensors:

- Delay all tensor contractions as long as possible

$$\text{e.g.} \qquad \tilde{k}_{bf} = \sum_{acdo\mu\nu} \mathcal{C}_{ao} \; c^{\mu}_{ab} I_{\mu\nu} c^{\nu}_{cd} \; \mathcal{C}_{co} \mathcal{C}_{df}$$

- Compare possible contraction schemes by complexity
- Execute cheapest evaluation scheme
- May incorporate and exploit symmetry

$\Rightarrow$ Determine optimal contraction sequence automatically

Acknowledgements

- Dr. James Avery
- Prof. Andreas Dreuw and the Dreuw group

- Prof. Guido Kanschat
- HGS Mathcomp

Questions?

- EMail: `michael.herbst@iwr.uni-heidelberg.de`
- Website/blog: `https://michael-herbst.com`
- Projects: `https://lazyten.org` and `https://molsturm.org`