Automatic Generation of Computer Codes for Correlated Wavefunction Calculations

Anastasios Papadopoulos, Hang Xu, Frank Neese

June 6, 2023

Max-Planck-Institut für Kohlenforschung

Aim of Quantum Chemistry

Solve Schrödinger equation for molecular (many-body) system

$$\hat{H}|\Psi\rangle = E|\Psi\rangle \tag{1}$$

Aim of Quantum Chemistry

Solve Schrödinger equation for molecular (many-body) system

$$\hat{H}|\Psi\rangle = E|\Psi\rangle \tag{1}$$

Why?

- Powerful tool to predict and support spectroscopic data
- Gain insight into the electronic structure and estimate reactivity

Aim of Quantum Chemistry

Solve Schrödinger equation for molecular (many-body) system

$$\hat{H}|\Psi\rangle = E|\Psi\rangle \tag{1}$$

Why?

- Powerful tool to predict and support spectroscopic data
- Gain insight into the electronic structure and estimate reactivity

No analytic solution for many-electron systems $\rightarrow$ find numerical solution

Finding a Solution to the Schrödinger Equation

Approximations

- Born-Oppenheimer $\rightarrow$ split nuclear and electronic coordinates

$$\hat{H}_{BO} = \hat{T}_e + \hat{V}_{ne} + \hat{V}_{ee} + V_n \tag{2}$$

- Variational solution can be described as a linear combination of Slater determinants (antisymmetrized product of one-electron functions)
- Basis set expansion

Finding a Solution to the Schrödinger Equation

Approximations

- Born-Oppenheimer $\rightarrow$ split nuclear and electronic coordinates

$$\hat{H}_{BO} = \hat{T}_e + \hat{V}_{ne} + \hat{V}_{ee} + V_n \tag{2}$$

- Variational solution can be described as a linear combination of Slater determinants (antisymmetrized product of one-electron functions)
- Basis set expansion

Even with all these approximations, it is unfeasible to evaluate this equation by hand!

Software and Quantum Chemistry

Software allows us to calculate energies and properties of systems as large as proteins nowadays

Myoglobin (heme group removed), ~ 2.4k atoms

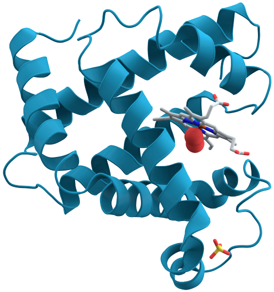
PBE

def2-SV(P), ~ 19k basis functions
def2-mTZVP/J, ~ 43k auxiliary basis functions

16 cores

100000 MB memory per core

total calculation: 3.5h (14 iterations)



Correlated Wavefunction Methods

For highly-accurate calculations, correlation methods are required, such as *configuration-interaction* (CI) or *coupled-cluster* (CC),

$$|\Psi_{CI}\rangle = (1 + \hat{C})\,|0\rangle \tag{3}$$

$$|\Psi_{CC}\rangle = e^{\hat{T}}\,|0\rangle \tag{4}$$

with $|0\rangle$ being the reference wavefunction, typically taken from a Hartree-Fock calculation (Slater determinant, mean-field solution). They aim to include *dynamical correlation*.

$$\hat{T} = \hat{T}_1 + \hat{T}_2 + \cdots + \hat{T}_N \tag{5}$$

$$= \sum_{ia} t_i^a a_a^\dagger a_i + \frac{1}{4}\sum_{ijab} t_{ij}^{ab} a_a^\dagger a_b^\dagger a_j a_i + \cdots \tag{6}$$

To reduce the cost of the calculation, these *Ansätze* are truncated. Most often only singles and doubles (CISD/CCSD) excitations are included. To improve the accuracy, higher-order excitations can be added.

Complexity of Quantum Chemical Methods

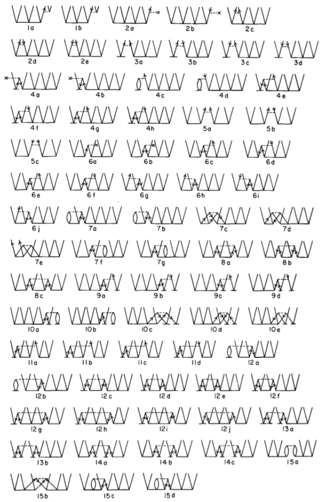
CC equations:

$$E_{CC} = \langle 0| e^{-\hat{T}} \hat{H} e^{\hat{T}} |0\rangle \tag{7}$$

$$0 = \langle \Phi_\mu | e^{-\hat{T}} \hat{H} e^{\hat{T}} |0\rangle \tag{8}$$

By including additional excitations, the equations become increasingly complex.

Complexity of Quantum Chemical Methods



All 74 diagrams contributing to T_4 in the CCSDTQ equations.[1]

[1]Kucharski, S. A.; Bartlett, R. J. *Theor. Chim. Acta* **1991**, *80*, 387–405.

Complexity of Quantum Chemical Methods

CC equations:

$$E_{CC} = \langle 0 | \, e^{-\hat{T}} \hat{H} e^{\hat{T}} \, | 0 \rangle \tag{7}$$

$$0 = \langle \Phi_\mu | e^{-\hat{T}} \hat{H} e^{\hat{T}} \, | 0 \rangle \tag{8}$$

By including additional excitations, the equations become increasingly complex.

Could take years to manually optimise those equations. The advantage is that they can be systematically improved.

Automated Generation

Systematic work is what computers do best.

Benefits

- Reduce implementation time
- Remove human-error (e.g. accidental sign flipping)
- Consistent implementations, reference for manual implementations
- Improvements to toolchain are easily transferred to all methods

[1]Kállay, M.; Surján, P. R. *J. Chem. Phys.* **2001**, *115*, 2945–2954.
[2]Hirata, S. *J. Phys. Chem. A* **2003**, *107*, 9887–9897.
[3]MacLeod, M. K.; Shiozaki, T. *J. Chem. Phys.* **2015**, *142*, 051103.
[4]Krupička, M. et al. *J. Comput. Chem.* **2017**, *38*, 1853–1868.
[5]Evangelista, F. A. *J. Chem. Phys.* **2022**, *157*, 064111.

Automated Generation

Systematic work is what computers do best.

Benefits

- Reduce implementation time
- Remove human-error (e.g. accidental sign flipping)
- Consistent implementations, reference for manual implementations
- Improvements to toolchain are easily transferred to all methods

Already succesfully applied to many methods, such as CI, CC, MBPT, both single- and multi-reference.[1,2,3,4,5]

[1]Kállay, M.; Surján, P. R. *J. Chem. Phys.* **2001**, *115*, 2945–2954.
[2]Hirata, S. *J. Phys. Chem. A* **2003**, *107*, 9887–9897.
[3]MacLeod, M. K.; Shiozaki, T. *J. Chem. Phys.* **2015**, *142*, 051103.
[4]Krupička, M. et al. *J. Comput. Chem.* **2017**, *38*, 1853–1868.
[5]Evangelista, F. A. *J. Chem. Phys.* **2022**, *157*, 064111.

Defining an automated toolchain

Formulate general protocol. Example: arbitrary-order coupled cluster

1. Define your *Ansatz* (input!)

$$|\Psi_{CC}\rangle = e^{\hat{T}}|0\rangle \tag{9}$$

$$\hat{T} = \sum_{ia} t_i^a a_a^\dagger a_i + \frac{1}{4}\sum_{ijab} t_{ij}^{ab} a_a^\dagger a_b^\dagger a_j a_i + \cdots \tag{10}$$

Defining an automated toolchain

Formulate general protocol. Example: arbitrary-order coupled cluster

1. Define your *Ansatz* (input!)

$$|\Psi_{CC}\rangle = e^{\hat{T}}|0\rangle \tag{9}$$

$$\hat{T} = \sum_{ia} t_i^a a_a^\dagger a_i + \frac{1}{4}\sum_{ijab} t_{ij}^{ab} a_a^\dagger a_b^\dagger a_j a_i + \cdots \tag{10}$$

2. Formulate your equations in a general fashion

$$\sigma_{ij\ldots}^{ab\ldots} = \langle \Phi_{ij\ldots}^{ab\ldots}| e^{-\hat{T}} \hat{H} e^{\hat{T}} |0\rangle \tag{11}$$

$$E_{CC} = \langle 0| e^{-\hat{T}} \hat{H} e^{\hat{T}} |0\rangle \tag{12}$$

$$e^{-\hat{T}} \hat{H} e^{\hat{T}} = \hat{H} + [\hat{H}, \hat{T}] + [[\hat{H}, \hat{T}], \hat{T}] + \ldots \tag{13}$$

Defining an automated toolchain

Formulate general protocol. Example: arbitrary-order coupled cluster

1. Define your *Ansatz* (input!)

$$|\Psi_{CC}\rangle = e^{\hat{T}}|0\rangle \tag{9}$$

$$\hat{T} = \sum_{ia} t_i^a a_a^\dagger a_i + \frac{1}{4}\sum_{ijab} t_{ij}^{ab} a_a^\dagger a_b^\dagger a_j a_i + \cdots \tag{10}$$

2. Formulate your equations in a general fashion

$$\sigma_{ij\ldots}^{ab\ldots} = \langle \Phi_{ij\ldots}^{ab\ldots}|e^{-\hat{T}}\hat{H}e^{\hat{T}}|0\rangle \tag{11}$$

$$E_{CC} = \langle 0|e^{-\hat{T}}\hat{H}e^{\hat{T}}|0\rangle \tag{12}$$

$$e^{-\hat{T}}\hat{H}e^{\hat{T}} = \hat{H} + [\hat{H},\hat{T}] + [[\hat{H},\hat{T}],\hat{T}] + \ldots \tag{13}$$

3. Generate equations

Defining an automated toolchain

Formulate general protocol. Example: arbitrary-order coupled cluster

1. Define your *Ansatz* (input!)

$$|\Psi_{CC}\rangle = e^{\hat{T}}|0\rangle \tag{9}$$

$$\hat{T} = \sum_{ia} t_i^a a_a^\dagger a_i + \frac{1}{4}\sum_{ijab} t_{ij}^{ab} a_a^\dagger a_b^\dagger a_j a_i + \cdots \tag{10}$$

2. Formulate your equations in a general fashion

$$\sigma_{ij\dots}^{ab\dots} = \langle \Phi_{ij\dots}^{ab\dots}| e^{-\hat{T}} \hat{H} e^{\hat{T}}|0\rangle \tag{11}$$

$$E_{CC} = \langle 0| e^{-\hat{T}} \hat{H} e^{\hat{T}}|0\rangle \tag{12}$$

$$e^{-\hat{T}} \hat{H} e^{\hat{T}} = \hat{H} + [\hat{H}, \hat{T}] + [[\hat{H}, \hat{T}], \hat{T}] + \dots \tag{13}$$

3. Generate equations

4. Process equations

Defining an automated toolchain

Formulate general protocol. Example: arbitrary-order coupled cluster

1. Define your *Ansatz* (input!)

$$|\Psi_{CC}\rangle = e^{\hat{T}}|0\rangle \tag{9}$$

$$\hat{T} = \sum_{ia} t_i^a a_a^\dagger a_i + \frac{1}{4}\sum_{ijab} t_{ij}^{ab} a_a^\dagger a_b^\dagger a_j a_i + \cdots \tag{10}$$

2. Formulate your equations in a general fashion

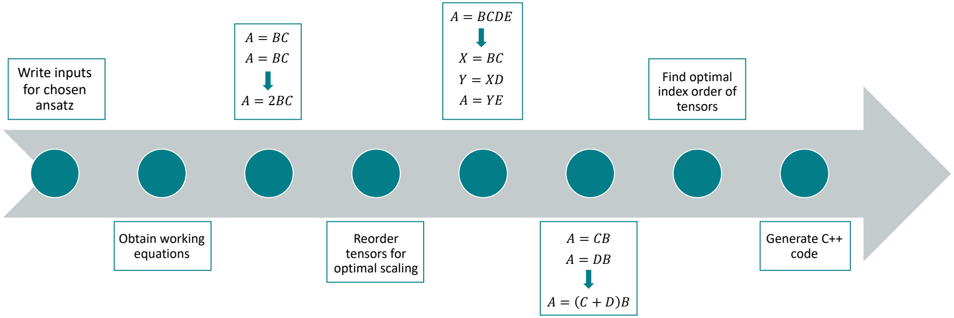
$$\sigma_{ij\ldots}^{ab\ldots} = \langle\Phi_{ij\ldots}^{ab\ldots}|e^{-\hat{T}}\hat{H}e^{\hat{T}}|0\rangle \tag{11}$$

$$E_{CC} = \langle 0|e^{-\hat{T}}\hat{H}e^{\hat{T}}|0\rangle \tag{12}$$

$$e^{-\hat{T}}\hat{H}e^{\hat{T}} = \hat{H} + [\hat{H},\hat{T}] + [[\hat{H},\hat{T}],\hat{T}] + \ldots \tag{13}$$

3. Generate equations
4. Process equations
5. Transform equations into source code

Meet the Automated Generator Environment (AGE)

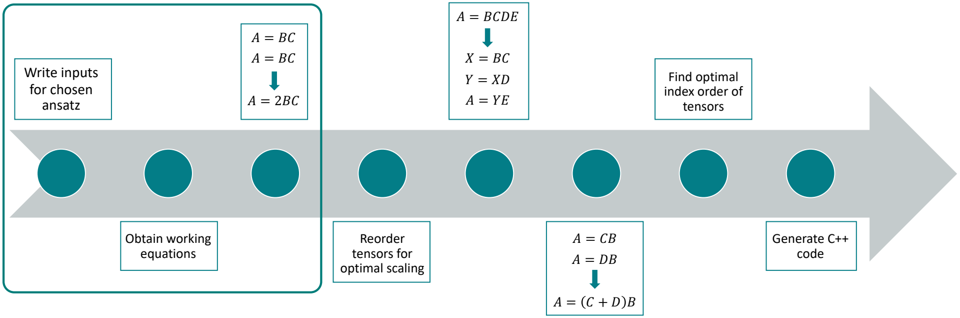


Modular toolchain

- Rewritten in C++ for increased performance
- Backend of math utilities (definition for indices, tensors, contractions, symmetry, etc.)
- Interface is a single equation (`.eq`) file

Meet the Automated Generator Environment (AGE)

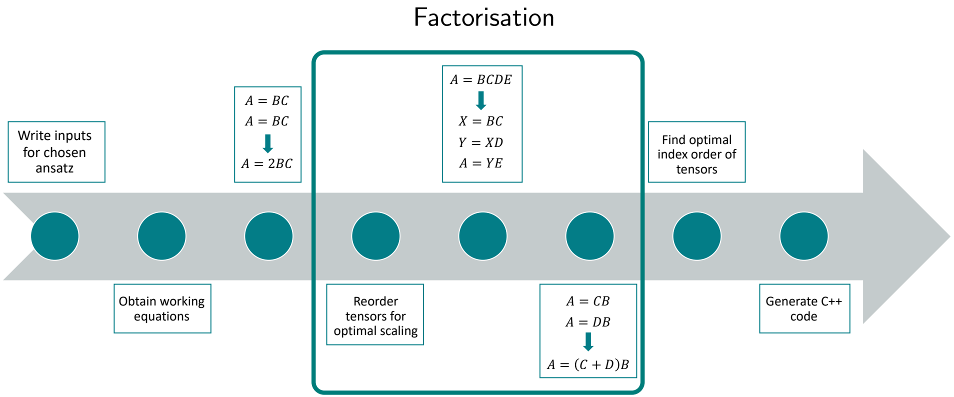
Equation generation



Modular toolchain

- Rewritten in C++ for increased performance
- Backend of math utilities (definition for indices, tensors, contractions, symmetry, etc.)
- Interface is a single equation (`.eq`) file

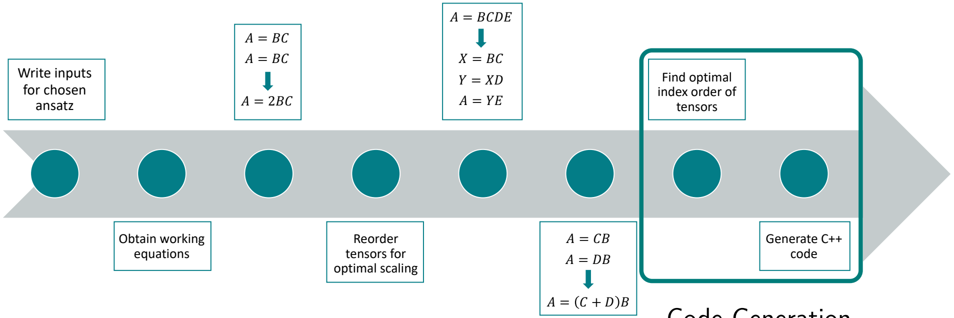
Meet the Automated Generator Environment (AGE)



Modular toolchain

- Rewritten in C++ for increased performance
- Backend of math utilities (definition for indices, tensors, contractions, symmetry, etc.)
- Interface is a single equation (`.eq`) file

Meet the Automated Generator Environment (AGE)



Code Generation

Modular toolchain

- Rewritten in C++ for increased performance
- Backend of math utilities (definition for indices, tensors, contractions, symmetry, etc.)
- Interface is a single equation (`.eq`) file

`AGE` Interface – `.eq` file

Once the equations are generated, we need some way to communicate these to the rest of the toolchain.

`.eq` file

- `AlcompHeader` - contains information regarding tensor storage, permutational symmetry, etc.
- Equations
  - $i, j, k, \ldots$ denote occupied indices
  - $a, b, c, \ldots$ denote virtual indices
  - $t, u, v, \ldots$ denote active indices
  - Summed indices are denoted by capitalised letters

`AGE` Interface – `.eq` file

Once the equations are generated, we need some way to communicate these to the rest of the toolchain.

`.eq` file

- `AlcompHeader` - contains information regarding tensor storage, permutational symmetry, etc.
- Equations
  - $i, j, k, \dots$ denote occupied indices
  - $a, b, c, \dots$ denote virtual indices
  - $t, u, v, \dots$ denote active indices
  - Summed indices are denoted by capitalised letters

$$S_{ij}^{ab} \leftarrow -\sum_{kc} (ki|bc)\tau_{kj}^{ac} \tag{14}$$

```
Sijab(a0,i0,b0,j0) += -1.0 I(K0,i0,b0,C0) Tau(a0,K0,C0,j0)
```

Equation Generation - Commutators

Use commutators to change the order of operators

$$E_q^p = a_{p\alpha}^\dagger a_{q\alpha} + a_{p\beta}^\dagger a_{q\beta} \tag{15}$$

$$[E_q^p, E_s^r] = E_s^p \delta_{rq} - E_q^r \delta_{ps} \tag{16}$$

$$E_p^i \left|\Phi_0\right\rangle = 2\delta_{ip} \left|\Phi_0\right\rangle, \quad \left\langle\Phi_0\right| E_i^p = 2\delta_{ip} \left\langle\Phi_0\right| \tag{17}$$

$$E_a^p \left|\Phi_0\right\rangle = 0, \quad \left\langle\Phi_0\right| E_p^a = 0 \tag{18}$$

Equation Generation - Commutators

Use commutators to change the order of operators

$$E^p_q = a^\dagger_{p\alpha} a_{q\alpha} + a^\dagger_{p\beta} a_{q\beta} \tag{15}$$

$$[E^p_q, E^r_s] = E^p_s \delta_{rq} - E^r_q \delta_{ps} \tag{16}$$

$$E^i_p \left|\Phi_0\right\rangle = 2\delta_{ip} \left|\Phi_0\right\rangle , \quad \left\langle\Phi_0\right| E^p_i = 2\delta_{ip} \left\langle\Phi_0\right| \tag{17}$$

$$E^p_a \left|\Phi_0\right\rangle = 0, \quad \left\langle\Phi_0\right| E^a_p = 0 \tag{18}$$

- Universally applicable (spin-free, spin-orbital)
- Simple
- Slow
- Redundant terms

Equation Generation – Wick's Theorem

`Wick&d`[6]

$$\overbracket{a_i^\dagger a_j} = \delta_{ij} \qquad \overbracket{a_a a_b^\dagger} = \delta_{ab} \tag{19}$$

$$\langle \Phi_i^a | \hat{F}_N | 0 \rangle = \sum_{pq} f_{pq} \langle 0 | \{ \overbracket{a_i^\dagger \overbracket{a_a a_p^\dagger} a_q} \} | 0 \rangle = f_{ai} \tag{20}$$

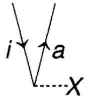
- Difficult to program
- Fast
- Redundant terms

[6]Evangelista, F. A. *J. Chem. Phys.* **2022**, *157*, 064111.

Equation Generation – Diagrams

MRCC[7]

$$\langle \Phi_i^a | \hat{F}_N | 0 \rangle =$$



$$= f_{ai} \qquad (21)$$

- Difficult to program
- Fast
- No redundant terms are generated

[7]Kállay, M.; Surján, P. R. *J. Chem. Phys.* **2001**, *115*, 2945–2954.

Equation Generation – Canonicalisation

Generating equations with commutators generates many redundant equations

$$E_{CC} \leftarrow \langle 0| \hat{H} \hat{T}_2 |0\rangle \tag{22}$$

Equation Generation – Canonicalisation

Generating equations with commutators generates many redundant equations

$$E_{CC} \leftarrow \langle 0| \hat{H} \hat{T}_2 |0\rangle \tag{22}$$

$$E_{CC} \leftarrow \sum_{ijab} (ia|jb)\, T_{ij}^{ab} \tag{23}$$

$$E_{CC} \leftarrow \sum_{ijab} (jb|ia)\, T_{ij}^{ab} \tag{24}$$

Equation Generation – Canonicalisation

Generating equations with commutators generates many redundant equations

$$E_{CC} \leftarrow \langle 0| \hat{H} \hat{T}_2 |0\rangle \tag{22}$$

$$E_{CC} \leftarrow \sum_{ijab} (ia|jb) T_{ij}^{ab} \tag{23}$$

$$E_{CC} \leftarrow \sum_{ijab} (jb|ia) T_{ij}^{ab} \tag{24}$$

These equations can be merged by utilising permutational symmetry

$$E_{CC} \leftarrow 2 \sum_{ijab} (ia|jb) T_{ij}^{ab} \tag{25}$$

Factorisation – Factorizer

- Factorize in terms of binary contractions
- Ensures formal scaling, such as $\mathcal{O}(N^6)$ for CCSD
- Identifies intermediates
- Finds "best" possible intermediates and contraction order
- Identifies common intermediates

Factorisation – Factorizer

- Factorize in terms of binary contractions
- Ensures formal scaling, such as $\mathcal{O}(N^6)$ for CCSD
- Identifies intermediates
- Finds "best" possible intermediates and contraction order
- Identifies common intermediates

Given the following contraction,

$$A = BCDEF, \tag{26}$$

this can be factorised in several ways:

$$A = (((BC)D)E)F \tag{27}$$

$$A = B((CD)(EF)) \tag{28}$$

$$A = (BC)(D(EF)) \tag{29}$$

$$\dots$$

Factorisation – Factorizer

- Factorize in terms of binary contractions
- Ensures formal scaling, such as $\mathcal{O}(N^6)$ for CCSD
- Identifies intermediates
- Finds "best" possible intermediates and contraction order
- Identifies common intermediates

Given the following contraction,

$$A = BCDEF, \tag{26}$$

this can be factorised in several ways:

$$A = (((BC)D)E)F \tag{27}$$

$$A = B((CD)(EF)) \tag{28}$$

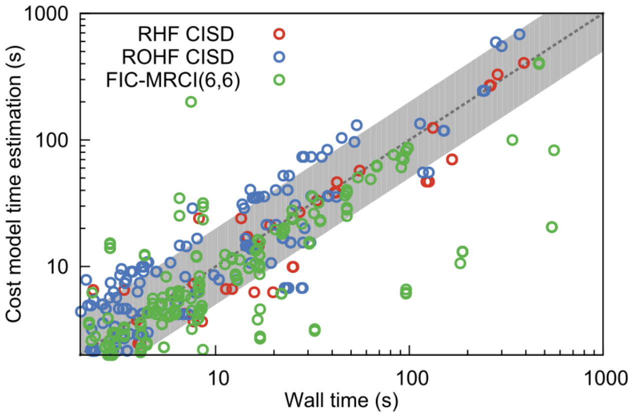
$$A = (BC)(D(EF)) \tag{29}$$

$$\dots$$

Pick the best one according to the *cost model*

Cost Model

In order to find the best possible intermediates and factorization, we need to have an estimate how long each contraction should take



Cost Model

Heuristic model that determines the FLOP count for a given contraction based on index space sizes

Cost Model

Heuristic model that determines the FLOP count for a given contraction based on index space sizes

Example:

$$E_{(T)} \leftarrow \sum_{ijkabc} t_i^a t_{ijk}^{abc} (jb|kc) \tag{30}$$

$$X_{jk}^{bc} = \sum_{ia} t_i^a t_{ijk}^{abc} \tag{31}$$

$$E_{(T)} \leftarrow \sum_{jkab} X_{jk}^{bc} (jb|kc) \tag{32}$$

$$X_i^a = \sum_{jkbc} t_{ijk}^{abc} (jb|kc) \tag{33}$$

$$E_{(T)} \leftarrow \sum_{ia} t_i^a X_i^a \tag{34}$$

Cost Model

Heuristic model that determines the FLOP count for a given contraction based on index space sizes

Example:

$$E_{(T)} \leftarrow \sum_{ijkabc} t_i^a t_{ijk}^{abc} (jb|kc) \tag{30}$$

$$X_{jk}^{bc} = \sum_{ia} t_i^a t_{ijk}^{abc} \tag{31}$$

$$E_{(T)} \leftarrow \sum_{jkab} X_{jk}^{bc} (jb|kc) \tag{32}$$

FLOP $= 6.402 \cdot 10^{10}$

$$X_i^a = \sum_{jkbc} t_{ijk}^{abc} (jb|kc) \tag{33}$$

$$E_{(T)} \leftarrow \sum_{ia} t_i^a X_i^a \tag{34}$$

FLOP $= 6.400 \cdot 10^{10}$

Factorisation – Distributive Law

Reduce prefactor for method, $\mathcal{O}(xN^y)$, by applying the *distributive law*

$$S \leftarrow AC \tag{35}$$

$$S \leftarrow BC \tag{36}$$

Factorisation – Distributive Law

Reduce prefactor for method, $\mathcal{O}(xN^y)$, by applying the *distributive law*

$$S \leftarrow AC \tag{35}$$

$$S \leftarrow BC \tag{36}$$

Addition is cheaper than multiplication

$$D \leftarrow A + B \tag{37}$$

$$S \leftarrow DB \tag{38}$$

I/O Minimisation for Intermediates

By default, all rank-4 tensors are stored on disk to reduce memory usage

$$t_{ij}^{ab} \rightarrow [t^{ab}]_{ij} \tag{39}$$

However, disk I/O is rougly 10-100 times slower than RAM, so we must minimise it to retain performance

I/O Minimisation for Intermediates

By default, all rank-4 tensors are stored on disk to reduce memory usage

$$t_{ij}^{ab} \rightarrow [t^{ab}]_{ij} \tag{39}$$

However, disk I/O is rougly 10-100 times slower than RAM, so we must minimise it to retain performance

Example:

$$[\sigma^{ab}]_{ij} \leftarrow [X^{ij}]_{ab} \tag{40}$$

```
for each i:
    for each j:
        load matrix Sij // a x b
        for each a:
            for each b:
                load matrix Xab // i x j
                Sij(a,b) += Xab(i,j)
        store matrix Sij
```

I/O Minimisation for Intermediates

By default, all rank-4 tensors are stored on disk to reduce memory usage

$$t_{ij}^{ab} \rightarrow [t^{ab}]_{ij} \tag{39}$$

However, disk I/O is rougly 10-100 times slower than RAM, so we must minimise it to retain performance

Example:

$$[\sigma^{ab}]_{ij} \leftarrow [X^{ab}]_{ij} \tag{41}$$

```
for each i:
    for each j:
        load matrix Sij // a x b
        load matrix Xij // a x b
        Sij += Xij      // entire matrix copy
        store matrix Sij
```

Code Generator

Once all equations have been factorised, the `generator` translates equations into `ORCA` source code

1. Determine I/O-minimal loop order for each contraction (cost model)
2. Load quantities from disk
3. If possible, apply hand-written functions to evaluate contractions with batching or use only BLAS
4. Otherwise, generate "naive" contraction code (explicit loops and element-wise access)
5. Package generated code into a module that can easily be interfaced with `orca_autoci`

Anytime an improvement has been made in the `AGE`, all old modules can easily be updated to the newer version

Code Generator – Resorting of 4-index tensors

One such improvement was on-the-fly resorting of 4-index "matrix containers" to enable more BLAS operations (resorted containers are stored on disk)

$$[\Gamma^{bi}]_{ia} \leftarrow \sum_j C_j^b C_j^a \tag{42}$$

Code Generator – Resorting of 4-index tensors

One such improvement was on-the-fly resorting of 4-index "matrix containers" to enable more BLAS operations (resorted containers are stored on disk)

$$[\Gamma^{bi}]_{ia} \leftarrow \sum_j C_j^b C_j^a \tag{42}$$

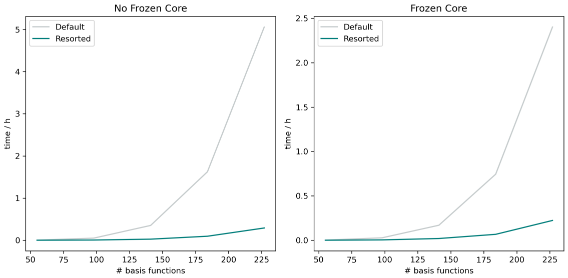
$$[\Gamma^{ba}]_{ii} \leftarrow \sum_j C_j^b C_j^a \tag{43}$$

Code Generator – Resorting of 4-index tensors

One such improvement was on-the-fly resorting of 4-index "matrix containers" to enable more BLAS operations (resorted containers are stored on disk)

$$[\Gamma^{bi}]_{ia} \leftarrow \sum_j C_j^b C_j^a \tag{42}$$

$$[\Gamma^{ba}]_{ii} \leftarrow \sum_j C_j^b C_j^a \tag{43}$$



RHF CISD 2RDM calculation, def2-TZVP, linear alkanes

Code Generator – TTGT

Known from literature as "Transpose-Transpose-DGEMM-Transpose"[8]

Any tensor contraction can be reformulated as a matrix multiplication,

$$C_{ij} = \sum_k A_{ik} B_{kj}, \tag{44}$$

in which compound indices i, j, k may refer to none, one, or multiple actual indices

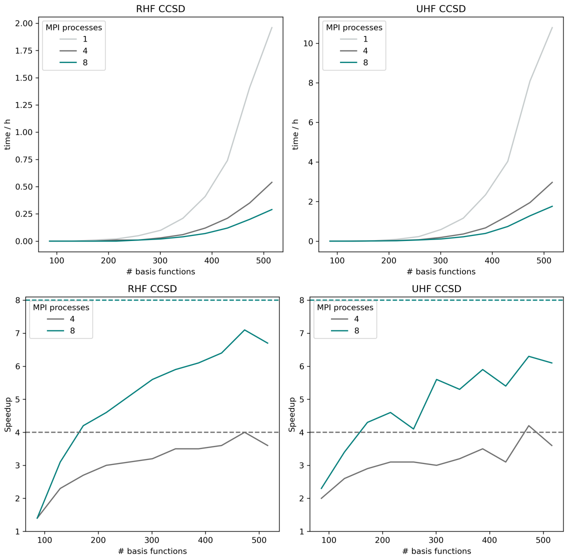
If BLAS is still not possible at this point, try resorting with the TTGT (except matrix containers) to enforce BLAS

[8]Springer, P.; Bientinesi, P. *ACM Trans. Math. Softw.* **2018**, *44*, 1–29.

Code Generator – Parallelised Code

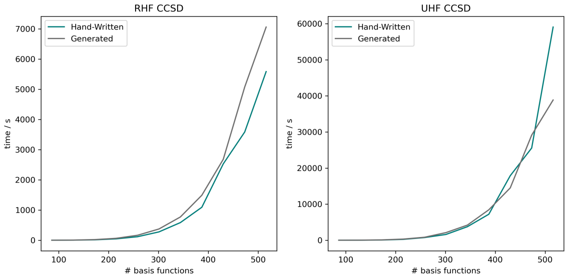
The `AGE`'s code generator is able to generate MPI parallelised code on a per-contraction basis

Code Generator – Parallelised Code



Average sigma iteration, def2-TZVP, linear alkenes

Comparison Generated and Hand-Written Code



Average sigma iteration, def2-TZVP, linear alkenes, serial

Generated code is competitive with the hand-written code!

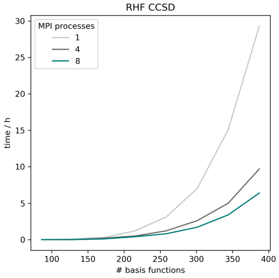
Correlation Methods Analytic Gradients

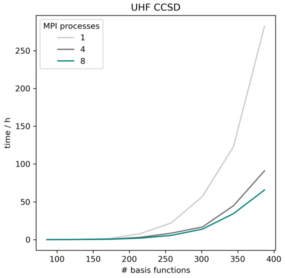
The starting point for deriving an analytic gradient for any non-variational method is to define a Lagrangian

$$\mathcal{L}_{CC} = \langle 0| (1+\hat{\Lambda}) e^{-\hat{T}} \hat{H} e^{\hat{T}} |0\rangle + \sum_{p>q} z_{pq} f_{pq} \tag{45}$$

The goal is to formulate that Lagrangian in terms of density matrices. At that point the rest of the gradient derivation is method independent. Only amplitude equations and density matrices need to be generated.

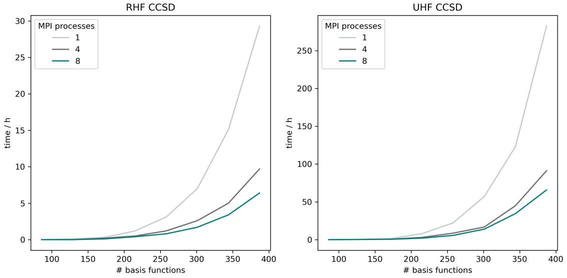
CC Analytic Gradients – Performance





Single gradient step, def2-TZVP, linear alkenes

CC Analytic Gradients – Performance



Single gradient step, def2-TZVP, linear alkenes

- CC gradients with 400 routinely achievable
- Decent parallel scaling
- Much faster than numerical gradients

Higher-Order Derivatives with the AGE

According to the $2n+1$ and $2n+2$ rules, perturbed equations need to be solved

- Symbolic level:
    1. Apply product rule and add perturbation labels to the wavefunction parameters (e.g. CC amplitudes) and integrals
    2. Split the LHS (perturbed amplitudes) with the RHS (perturbed integrals)
- Numerical level:
    1. Perturbed integrals (MO basis, relaxed & unrelaxed)

While generating the new set of equations, non-contributing terms are filtered out (e.g. perturbed integrals where the basis functions do not depend on the external perturbation)

Alternative Strategies for Generating Perturbed Equatior

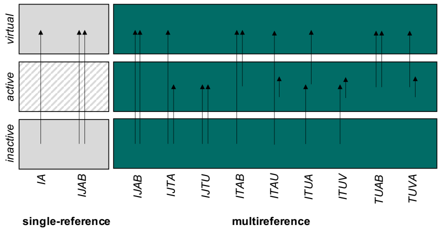
Instead of using the product rule and string manipulation, change the kernel

$$\hat{H}_{\text{eff}} = e^{-\hat{T}}\hat{H}e^{\hat{T}} = (\hat{H}e^{\hat{T}})_{\text{C}} \tag{46}$$

$$\begin{aligned}\hat{H}^{\chi}_{\text{eff}} =& \hat{H}^{\chi} + [\hat{H}, \hat{T}]^{\chi} + \frac{1}{2!}[[\hat{H}, \hat{T}], \hat{T}]^{\chi} \\ &+ \frac{1}{3!}[[[\hat{H}, \hat{T}], \hat{T}], \hat{T}]^{\chi} + \frac{1}{4!}[[[[\hat{H}, \hat{T}], \hat{T}], \hat{T}], \hat{T}]^{\chi} + \dots \\ =& (\hat{H}^{\chi}e^{\hat{T}})_{\text{C}} + ([\hat{H}, \hat{T}^{\chi}]e^{\hat{T}})_{\text{C}}\end{aligned} \tag{47}$$

This way we can already screen non-contributing terms in cases where tensors do not depend on the perturbation

fic-MRCI/MRCC



fic-MRCI:

$$E_{MRCI} = \langle 0 | \hat{H} | \Psi_{MRCI} \rangle \tag{48}$$

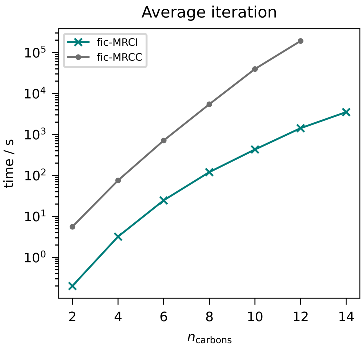
$$0 = \langle \Phi_\mu | (\hat{H} - E) | \Psi_{MRCI} \rangle \tag{49}$$

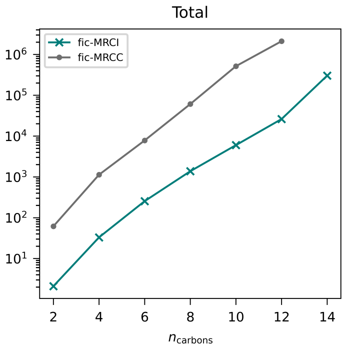
fic-MRCC:

$$E_{MRCC} = \langle 0 | e^{-\hat{T}} \hat{H} e^{\hat{T}} | 0 \rangle \tag{50}$$

$$0 = \langle \Phi_\mu | \hat{H} + [\hat{H}, \hat{T}] + \frac{1}{2}[[\hat{H}, \hat{T}], \hat{T}] | 0 \rangle \tag{51}$$

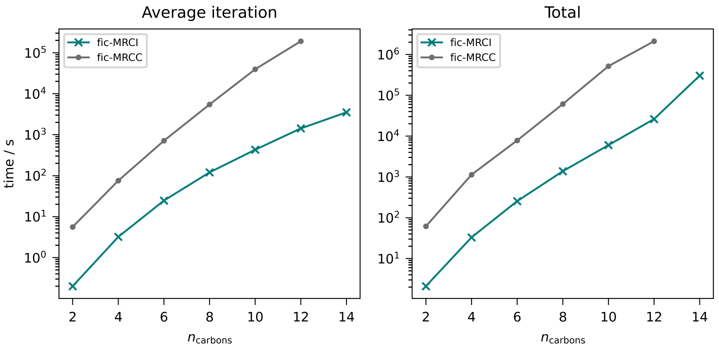
fic-MRCI/MRCC – Performance





Growing polyenes calculations, def2-SVP, CAS($n_{carbons}$, $n_{carbons}$)

fic-MRCI/MRCC – Performance



Growing polyenes calculations, def2-SVP, CAS($n_{carbons}$, $n_{carbons}$)

Multireference calculations with CAS(14,14) are a reality!

Conclusion

A general code generation toolchain has been built that generates competitive, consistent and parallelised code

CI/CC gradients have been achieved in `ORCA` using the `AGE`, which can be used for routine calculations of systems with 400 basis functions

Able to generate 2nd order derivatives

Multi-reference correlation calculations have been performed succesfully on systems with a CAS(14,14) space

Future of Automated Generation

Code generation will play an important role in future quantum chemistry. It enables us to implement "impossibly complicated" theories.

In the future, we could only keep a wavefunction *Ansatz* in the repository and generate the code during compile time. All improvements are immediately transferred to the program.

Acknowledgements

Hang Xu

Kantharuban Sivalingam

Ute Becker

Marvin Lechner

`AGE` team

Frank Neese