UNIVERSITY OF COPENHAGEN

Massively lockstep-parallel algorithms for full-isomer space quantum chemistry

subtitle

Masters

Anton M. Nielsen & Asmus Tørsleff

Supervised by <James title> James Avery
Co-supervised by Professor, Ph.D., Dr. Scient. Kurt V. Mikkelsen

Department of Computer Science
Department of Quantum Information Science

August 15, 2025

Masters

Massively lockstep-parallel algorithms for full-isomer space quantum chemistry

By Anton M. Nielsen & Asmus Tørsleff

Supervised by <James title> James Avery
Co-supervised by Professor, Ph.D., Dr. Scient. Kurt V. Mikkelsen

Date of submission: August 15, 2025

University of Copenhagen
Faculty of Science
Department of Computer Science
Department of Quantum Information Science
Universitetsparken 5
2100 Copenhagen Ø

Acknowledgements

I would like thank my supervisor Professor, Ph.D., Dr. Scient. Kurt V. Mikkelsen for guidance and advise during my masters degree.

I would like to thank Kurts research group for their discussions and advice.

I would like to thank my friends and family for supporting me and reminding me of a world outside of chemistry.

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Contributions to papers

This masters thesis is partly based on the following paper which is attached in the articles appendix.

In the paper "**??**", Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Table of Contents

I Thesis **1**

0.1 Introduction 1
0.2 Theory 2
  0.2.1 High Performance Parallel Computing 2
0.3 Methodology 6
  0.3.1 Testing 6
0.4 Code Structure 13
0.5 Extended Hückel Theory Matrix for GFN2-xTB 14
0.6 Fock Matrix for GFN2-xTB 15
  0.6.1 Isotropic Electrostatic and Exchange-correlation contribution 16
  0.6.2 Anisotropic Electrostatic and Exchange-correlation contribution 16
  0.6.3 Dispersion contribution 19
0.7 Total Energy for GFN2-xTB 22
  0.7.1 Repulsion Energy 22
  0.7.2 Extended Hückel Theory Energy 23
  0.7.3 Isotropic electrostatic and Exchange-correlation energy 23
  0.7.4 Anisotropic electrostatic energy 24
  0.7.5 Anisotropic XC energy 24
  0.7.6 Dispersion Energy 25
  0.7.7 SAD - Superposition of Atomic Densities 26
0.8 AI Declaration 27

II Appendicies **28**

A An appendix 29

Bibliography **30**

III Articles **31**

Part I

Thesis

0.1 Introduction

As part of James E. Avery's efforts to develop an efficient screening pipeline for fullerenes and potentially fulleroids we will in this report detail our efforts in porting parts of the xtb program by Grimme et al to SYCL code. The goal is a highly optimised and fully lockstep-parallel implementation of the electronic structure calculations from the GFN2-xTB method. A previous thesis by la Cour provides an efficient lockstep-parallel implementation of a forcefield method for computing geometric structures of fullerenes, which we will take to be our input.

The mentioned screening pipeline would enable the search of entire isomer spaces for fullerenes with certain properties such as a low lowest energy state indicating a stable isomer.

A fullerene is a molecule consisting only of carbon atoms connected in 12 pentagons and enough hexagons to create a hollow structure. As we increase the number of atoms the isomer space quickly grows leading to very slow search times. We aim to provide a quick and relatively accurate method for discarding large unpromising parts of the isomer space before searching with more accurate methods.

Fulleroids are essentially an extension to fullerenes where we allow n-gons instead of only penta- and hexagons, as long as we can still create a closed shape.

After a literature review we settled on the Geometry, Frequency, Noncovalent, extended Tight Binding (GFNn-xTB) family of methods as they are relatively accurate and quite fast at predicting electronic structures to a reasonable accuracy. We hope that this inherent speed will aid in getting good though-put after the transformation to a lockstep-parallel version. The GFNn-xTB metods come in iterations. GFN1 is the first and lays the ground work for the later iterations. It does however rely on element pairwise specific constants. In GFN2 this has been changed in favour of only element specific parameters. GFN0 is a more approximate and faster version of GFN2. And GFN-FF takes this trade-off further as this is a forcefield method which is parametrised using the insights (and parameters) gained from the other GFN iterations.

Forcefield methods save on computing all the pairwise interactions between atoms in a molecule and instead use efficient rules to lump atoms together in predictable clumps which then interact with other clumps. This can save tremendous effort.

Specifically GFN2 seemed most promising for our purposes as it is more accurate than GFN0 and simpler than GFN1, and if it is not fast enough would be relatively easy to then implement GFN0. GFN-FF was not considered suitable due to us wanting to see if it could be fast enough without defaulting to a forcefield method.

Lockstep-parallelisation is a paradigm best suited for GP-GPU. It takes advantage of the fact that GPUs operate more efficiently when all the cores are doing the same operations in a predictable fashion. This essentially is a step beyond data parallelism. We are not only operating on the same data across cores, but also doing the exact same steps. This means no conditionals with a data dependent evaluation. It is fine to have a loop that runs five times, opposed to say `data[coreId]` times.

0.2 Theory

0.2.1 High Performance Parallel Computing

The xTB program uses a specified variant of the GFN-xTB algorithm to compute various energies for a molecule. By default the program is set to use GFN2-xTB, which is also the variant this project focuses on.

For a molecule on the smaller scale such as a caffeine molecule which has 24 atoms, the time it takes to run xTB is perceivably fairly instantanious. Even a fullerene with 200 carbon atoms takes just about 2 seconds to compute on average on a 12th gen intel mobile processor. The computional time for even small molecules begins to be noticable, when the problem size grows to thousands or millions of molecules.

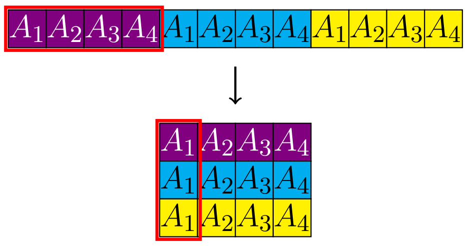
This project aims to compute the energies of fullerenes in the isomerspaces $C_{20}, \cdots, C_{200}$, and for this purpose the computational time needed for individual molecules is of less importance. What is truly interesting for this problem domain is speeding up largely concurrent xTB computations by running them in parallel on general purpose graphic processing units (GPGPUs).

The highest level of parallelization here is to compute all the energy terms of a molecule in the same kernel. There are no data dependencies between the computations of multiple molecules, and this makes it a perfect case for massive parallelization by distributing these isolated workloads across the thousands of threads supported on modern GPGPUs. Within the area of computing, the idea of running the same operations in parallel is known as a lockstep system. With a focus on fullerenes, which consists exclusively of carbon atoms, this type of lockstep parallelization is exactly what we want to create a fast and constant flow of data for the broader pipeline that this project is part of.

Since the executions of the xTB algorithm on each fullerene are completely isolated workloads, this means that the level of parallelization for a given isomer group, such as C20, scales with the amount of isomers in that group. This means that a much larger isomer group like C200 will also have a much greater level of parallelization.

The streaming multiprocessors (SMs) on a GPU are slower and simpler than the cores on a CPU. They have no branch prediction or other smart optimization techniques, but instead an SM has more threads it can execute in parallel in comparison to a CPU core which can only execute threads concurrently. The difference is that SMs can truly run its threads simultaniously, while CPU cores rely on context switching to make it seem like proccesses are running simultaniously. With SMs, working only on a few fullerenes will have a massive overhead from spinning up a kernel and copying data from the host(CPU) to the device(GPU), but the problem size for this project makes SMs a great fit.

The current Fortran implementation of the xTB program only takes a single molecule at a time, but when doing lockstep parallization it would be interesting to have an implementation that takes multiple molecules. This would avoid the overhead of starting multiple processes, and the program will have the data for all molecules, which gives opportunity for data coalescing by aligning the data as a structure of arrays (SOA) instead of and array of structures (AOS). It can also make copying data from the host to device more efficient since the data for multiple molecules can be moved together by the same instruction. To allow for coalesced access to the data on the device, we can essentially realign the data so that parts of the molecules that are accessed together, are also close together in memory. Doing this should also decrease the amount of copies needed between host and device memory, resulting in overall faster execution.



A_1 A_2 A_3 A_4 A_1 A_2 A_3 A_4 A_1 A_2 A_3 A_4
A_1 A_2 A_3 A_4
A_1 A_2 A_3 A_4
A_1 A_2 A_3 A_4

Figure 1.: An example of aligning molecules in memory to allow for coalesced access of the data.

An example of turning an AOS into a SOA can be seen in Figure 1 where the data is realigned to allow for coalesced data access. If the code accesses only a few atoms of a molecule at a time, then it is a waste if the memory page contains the rest of a molecule as well. Instead of copying whole molecules at a time, the data should be structured such that only the required data of a molecule is copied as it allows the kernel to get that data for more molecules at a time, thus reducing unnecessary copies and speeding up the computation.

The types of memory typically found in the memory hiearchy on a GPU are global, shared, local, and register memory. Using these levels of memory properly is crucial for achieving high performance in parallel computing tasks. Here is an overview of the various levels:

- Global - Accessible from all threads on the GPU. This is the largest but also the slowest pool of memory.
- Shared - Tied to a thread block (or workgroup), so it can be accessed by the threads in the same thread block. This pool of memory is smaller but faster than global memory.
- Register Memory - Each thread on a GPU has private access to a number of registers. This is the fastest type of memory used to store local variables and intermediate results.
- Local Memory - Also private to each thread. This type is slower than register memory and is usually used when there is insufficient space for variables on the registers of a thread.

The NVIDIA Ada GPU architecture[**nvidia-ada-tuning-guide**] has 64,000 32-bit registers per SM. This gives us 256 KB of register memory.

$$\begin{aligned} 64000 \cdot 32 &= 2048000 \text{ bits} \\ &= 256000 \text{ bytes} \\ &= 256 \text{ KB} \end{aligned} \tag{0.1}$$

Each thread has a maximum of 255 registers, and each SM has a maximum of 24 thread blocks. An SM has a maximum of 48 warps, so full utilization can be achieved when each thread block has 2 warps allocated. A warp has 32 threads, so an SM has a total of 1536 threads. Distributing the 64,000 registers evenly over these threads gives each thread 41 registers, which is considerably lower than the maximum of 255. This configuration allows utilizing 251.9 KB of the register memory available to the SMs.

$$\begin{aligned} 24 \cdot 2 \cdot 32 \cdot 41 \cdot 32 &= 2015232 \text{ bits} \\ &= 251904 \text{ bytes} \\ &= 251.904 \text{ KB} \end{aligned} \tag{0.2}$$

This gives each thread in an SM 164 bytes of register memory to work with.

$$\frac{251904}{1536} = 164 \text{ bytes} \tag{0.3}$$

The shared memory capacity per SM is 100 KB and a single thread block can have a maximum of 99 KB of this. This gives each thread block about 4.16 KB of shared memory, meaning that the 64 threads in a thread block will have 65 bytes each.

83GB for 10000 C200 8.3 MB for each molecule SMs has max 48 warps each warp has 32 threads 64000 32bit registers per SM each thread can max use 255 registers max thread blocks per SM is 24 two thread blocks to make use of all 48 warps shared memory capacity per SM 100KB maximum shared memory per thread block is 99KB SMs schedule warps/workgroups/thread blocks we can do 1536 molecules per SM with this amount of data

Doing more fine grained parallelization can make use of SIMT(Single instruction multiple threads) like reduction.

0.3 Methodology

0.3.1 Testing

The xTB algorithm computes a lot of different energies, corrections, and other additions to the energy terms. There is a lot of overlap between the different variants of xTB, so even focusing on just one of them still requires a great amount of code. Dealing with large computations with so many small parts makes proper validation especially important, as it becomes increasingly easy to make mistakes. In the case of the xTB algorithm, the original Fortran implementation becomes an important point of reference for comparison. Throughout the coding process it became apparent that the xTB implementation by the Grimme research group does not match the equations described in the original xTB paper by the same authors.

With this realization the obvious approach forward was to lean towards the existing implementation rather than the paper. This choice would allow us to continue doing validation against the existing code as a reference.

One of the hurdles from testing against an existing program becomes the lack of transparency regarding the logic that takes place between the initial input and the final output presented to the user. Thankfully, the source code is publicly available allowing for easy manipulation of the original flow of execution, thus avoiding the hassle of testing against a black box or the need to resort to methods of reverse engineering.

On behalf of these considerations it was decided to write patches that allow to intercept the arguments and results of arbitrary functions just by running the program as normal. This meant that smaller parts could be implemented without the need to implement all the code needed to compute its arguments.

An important part of any software is reproducability, and applying certain patches in certain scenarios is something that should preferably be automated, reproducable, and optimally also portable. This is especially important for this approach to validation as it requires a way to reproduce a specific version of xTB linked to the same versions of dependencies. Essentially an exact copy of the original shell environment to ensure that patches work, results are the same, and no new bugs appear in the program itself or its dependencies. All of this should be achievable without having to add, remove, downgrade or upgrade system packages on your system.

The well-known contenders for this is any of the numerous containerization solutions on Linux, such as Docker, Podman, LXC etc. There are some problems with these options though, one being that it can be difficult to truly reproduce package versions without saving the resulting container image, another being that it does not solve the problem of having multiple versions of the same package installed. Some other notable limitations are that it limits the process to run within the container and passing in a GPU or other hardware can be nefariously difficult. A container also does not have access to the X or Wayland session needed to run GUI applications, though that is not currently relevant in this case.

Another approach which has been growing in popularity in recent years are tools that take unique approaches to package management in order to make not only packages reproducable, but also shell environments, system configurations and other forms of "outputs". Two such popular package managers are Nix from the Nix team and Guix from the GNU foundation. Nix is arguably the more popular option and it is also the solution that has been chosen for this project.

Nix is an umbrella term that can refer to either the Nix functional programming language, the package manager, or the Nix based Linux distribution NixOS. The language and package manager go hand in hand and can be used on any Linux distribution. As such, NixOS is not required for the needs of this project and will not be mentioned going forward.

Nix does not follow the Unix Filesystem Hierarchy Standard (FHS), which brings with it some challenges, but this fundamental difference from other package managers is a major part of what makes Nix so powerful. Rather than installing packages into the usual system paths like '/bin', '/lib' etc. Nix installs everything into a read-only path called the Nix store under '/nix/store'. Everything in the Nix store is a result of a core concept in Nix called a derivation, which is essentially a build task to produce some output of files into the Nix store. All outputs into the store is marked with a custom hash in the filename called a NAR hash. These fundamental ideas fix some common problems such as circular dependencies and allow having multiple versions of the same package installed as they will simply coincide in the Nix store with different NAR hashes.

The typical binary on Linux is dynamically linked against the FHS compliant paths and it is not uncommon to have them hardcoded either. To make use of the packages in the Nix store, it is required to either recompile the program against the store paths, or in the case of proprietary software, patching the ELF header is needed to change the path to the interpreter and to dynamically linked libraries. Thankfully the Nix package repository 'NixPkgs' is the largest and freshest out there[1], so as a typical user doing this is rarely needed. Nixpkgs is a version-controlled repository on GitHub, so using older versions of packages even alongside newer ones, is fairly trivial as it simply requires fetching multiple revision of the repository.

This along with the previously mentioned features have allowed a greatly simplified process of not only running the newest version 6.7.1 of the xTB program, but also running the much older nvfortran compatible version 6.4.0 alongside it. NixPkgs is also a collection of library functions, and the helper functions for making derivations called ‘mkDerivation‘ make it easy to define all the stages of packaging a program including unpacking, patching, building, checking, and installing the files. With this, the whole pipeline of patching, compiling, running, and passing the data over to the Python validation tests can be achieved with a single shell command.

```
nix run .#cmp-impls
```

This command takes the form ’`nix run <path to flake>#<output>`’. Path to flake refers to a file-tree whose root directory contains a file called ’flake.nix’. Nix flakes is an experimental but widely adopted feature, which provides a standard way to write Nix expressions and a way to manager their dependencies through a version-pinned lock file. The ’flake.nix’ file follows a uniform naming schema for declaring inputs and outputs, where inputs are the dependencies, and outputs are Nix expressions to be exposed. The new Nix command-line interface needed to interact with flakes is naturally also an experimental feature that has to be enabled explicitly. The run command instructs Nix to build and run the derivation ’cmp-impls’, which is defined as an app in the flake outputs.

[1]`https://repology.org/repositories/graphs`

```nix
{
  inputs = {
    nixpkgs.url = "github:nixos/nixpkgs/nixos-unstable";
  };

  outputs = { self, nixpkgs, ... }: {
    apps."x86_64-linux" = let
      pkgs = nixpkgs.legacyPackages."x86_64-linux";
      ...
    in {
      "cmp-impls" = let
        python = (pkgs.python3.withPackages (python-pkgs: with python-pkgs; [
          numpy scipy cvxopt
        ]));
      in {
        type = "app";
        program = toString (pkgs.writeShellScript "cmp-impls" ''
          PYTHONPATH=${pkgs.lib.cleanSource ./xtb-python} exec ${python}/bin/python \
            ${./xtb-python/cmp_impls.py} ${xtb_test_data}
        '');
      };
    };
    ...
  };
}
```

Python is declared with the required packages and is then used in the app to call the `cmp_impls.py` script. The script is called with the test data acquired from running the Fortran xTB program. This data comes from another derivation which executes patched versions of xTB and DFT-D4 on a C200 fullerene to get the relevant function arguments and results as binary files.

```nix
xtb_test_data = builtins.derivation {
  name = "xtb-test-data";
  system = "x86_64-linux";
  builder = "${pkgs.bash}/bin/bash";
  src = ./xtb-python/data/C200.xyz;
  args = ["-c" ''
    PATH=$PATH:${pkgs.coreutils}/bin
    mkdir -p ./calls/{build_SDQH0,coordination_number,\
      dim_basis,dtrf2,electro,form_product,get_multiints,\
      h0scal,horizontal_shift,multipole_3d,newBasisset, olapp}
    ${xtb}/bin/xtb $src
    ${dftd4}/bin/dftd4 $src
    mv calls $out
  ''];
};
```

The directories for the binary files are created in advance as checking whether they exist when writing the binary files has a large overhead. This derivation in turn uses derivations for xTB and DFT-D4. Luckily DFT-D4 is already in NixPkgs, but it still needs to be patched in order to extract the required data for validation. Thankfully the `mkDerivation` function used in NixPkgs makes overriding and patching a package very straightforward.

```
dftd4 = (pkgs.dftd4.overrideAttrs (finalAttrs: previousAttrs: {
  src = pkgs.fetchFromGitHub {
    owner = "dftd4";
    repo = "dftd4";
    rev = "502d7c59bf88beec7c90a71c4ecf80029794bd5e";
    hash = "sha256-FEABtBAZK0xQ1P/Pbj5gUuvKf8/ZLITXaXYB+btAY/8=";
  };
  buildInputs = [ multicharge ] ++ previousAttrs.buildInputs;
  doCheck = false;
  patches = previousAttrs.patches ++ [
    ./nix/patches/dftd4/use_gfn2.patch
    ./nix/patches/dftd4/log_args_and_outputs.patch
  ];
}));
```

The version is bumped by overriding the source, and the multicharge project is added from NixPkgs and also bumped as a requirement of this newer version. Some of the tests were timing out, so they have been disabled by setting `doCheck` to false. Lastly the patches are applied by providing the relevant patch files.

xTB and two of its dependencies, namely CPCM-X and numsa are not in NixPkgs and had to be packaged from scratch.

All the patches follow the structure below where the original function is prefixed with a 'g', such that the new wrapper function will be called instead. The wrapper function writes the function arguments to a binary file before calling the actual function and then finally writes the result to the same file. Writing a file for each call to a function is a bit excessive and will produce a very large amount of files, so a threshold has been used to create an upperbound on the number of files that can be created for each function.

```
+   logical :: hit_threshold
+   integer :: u
+   character(len=200) :: path
+
+   hit_threshold = testfile_path('electro', path)
+   if (.not.hit_threshold) then
+     open(newunit=u, file=trim(path), form='unformatted', access='stream')
+     write(u) nbf
+     write(u) size(H0), H0
+     write(u) size(P, 1), size(P, 2), P
...
+     if (allocated(ies%thirdOrder%atomicGam)) then
+       write(u) size(ies%thirdorder%atomicgam), ies%thirdorder%atomicgam
+     else
+       write(u) 0
+     end if
...
+     write(u) size(ies%jmat, 1), size(ies%jmat, 2), ies%jmat
+     write(u) size(ies%shift), ies%shift
+   end if
+
+   call gelectro(n,at,nbf,nshell,ies,H0,P,dq,dqsh,es,scc)
+
+   if (.not.hit_threshold) then
+     write(u) es
+     write(u) scc
+     close(u)
+   end if
```

When all the data has been written, then the binary files are passed onto `cmp-impls.py`, which is the test suite for comparing the Python reimplementation to the original Fortran code.

```python
def test_electro():
    fn_name = "electro"
    for i, file_path in enumerate(glob.glob(f'{directory}/{fn_name}/*.bin')):
        with open(file_path, 'rb') as f:
            def read_ints(n=1):
                return np.fromfile(f, dtype=np.int32, count=n)

            nbf = read_ints(1)[0]
            H01 = read_ints(1)[0]
            H0 = np.fromfile(f, dtype=np.float64, count=H01)
            m, n = read_ints(2)
            P = np.fromfile(f, dtype=np.float64, count=m * n).reshape((n, m))
            ...
            atomicGam1 = read_ints(1)[0]
            atomicGam = None  if atomicGam1 == 0
                              else np.fromfile(f, dtype=np.float64, count=atomicGam1)
            ...
            es_res, scc_res = read_reals(2)
            es, scc = electro(nbf, H0, P, dq, dqsh, atomicGam, shellGam, jmat, shift)

            is_equal(es, es_res, "es", fn_name)
            is_equal(scc, scc_res, "scc", fn_name)

    print(f"matches! [{fn_name}]")
```

0.4 Code Structure

```
xTB-math/
├─ bin2xyz/
│  ├─ bin2xyz.cpp ........................ converter from float64 coords into xyz format
│  └─ C200_10000_fullerenes.float64 ..................... 3d coords for 10k fullerenes
├─ flake.lock
├─ flake.nix .......................... conventional structure for Nix inputs and outputs
├─ nix/ ........................... Nix package definitions for xTB and its dependencies
│  ├─ cpx.nix ......................................... CPCM-X - a solvation model
│  ├─ numsa.nix ............................. solvent accessible surface area calculation
│  ├─ patches/ .............................. patches for extracting data for validation
│  │  ├─ dftd4/ ............................................. dispersion correction
│  │  │  ├─ log_args_and_outputs.patch
│  │  │  └─ use_gfn2.patch
│  │  └─ xtb/
│  │     ├─ log_args_and_outputs.patch
│  │     ├─ log_electro.patch
│  │     └─ log_utils.patch
│  └─ xtb.nix .................................... extended tight-binding program
├─ README.md
├─ report/
└─ xtb-python/ ............................. Python port of xTB paper and Fortran impl
   ├─ basisset.py
   ├─ blas.py
   ├─ cmp_impls.py ............................. validation tests against Fortran impl
   ├─ data/
   │  ├─ C200.xyz
   │  └─ caffeine.xyz
   ├─ dftd4.py ................................ computation for dispersion correction
   ├─ dftd4_reference.py ........................................ constants for dftd4
   ├─ energy.py ........................................ various energy computations
   ├─ fock.py .............................................. Fock matrix computation
   ├─ gfn2.py ........................................... xTB-GFN2 specific constants
   ├─ lapack.py ....................................... Facade for LAPACK functions
   ├─ scc.py ................................. computation for self-consistent charges
   ├─ slater.py ................................ computation for slater determinants
   ├─ util.py
   └─ xyz_reader.py
```

```
xtb-gpu/
├── flake.lock
├── flake.nix
├── nix/
│   ├── nvhpc.nix ........................ patching nvhpc to make nvfortran work on Nix
│   └── xtb.nix ................................ xtb version 6.4.0 compiled with nvfortran
├── README.md
├── sycl/
│   ├── build_SDQH0.cpp ......................... incomplete SYCL impl for build_SDQH0
│   └── data/ ............................. test data for electro.cpp computed from caffeine
│       ├── atomicGam.txt
│       ├── dqsh.txt
│       ├── dq.txt
│       ├── H0.txt
│       ├── jmat.txt
│       ├── P.txt
│       ├── shellGam.txt
│       └── shift.txt
└── electro.cpp ........................... SYCL impl for computing electrostatic energy
```

0.5 Extended Hückel Theory Matrix for GFN2-xTB

$$\begin{aligned} H^{EHT}_{\mu\nu} = \frac{1}{2} & K^{ll'}_{AB} S_{\mu\nu}(H_{\mu\mu} + H_{\nu\nu}) \\ & \cdot X(EN_A, EN_B) \\ & \cdot \Pi(R_{AB}, l, l') \\ & \cdot Y(\zeta^A_l, \zeta^B_{l'}), \forall \mu \in l(A), \nu \in l'(B) \end{aligned} \tag{0.4}$$

where μ and ν are AO indecies, l and l' index shells. Both AO's are associated with an atom labled A and B. $K^{ll'}_{AB}$ is a element and shell specific fitted constant however, in GFN2 it only depends on the shells. $S_{\mu\nu} = \langle\phi_\mu|\phi_\nu\rangle$ is just the overlap of the orbitals. In GFN2 $H_{\kappa\kappa} = h^l_A - \delta h^l_{CN'_A} CN'_A$ where CN'_A is the modified GFN2-type Coordinate Number for the element of atom A.

$$\begin{aligned} CN'_A = & \sum_{B \neq A}^{N_{\text{atoms}}} (1 + e^{-10(4(R_{A,\text{cov}} + R_{B,\text{cov}})/3R_{AB} - 1)})^{-1} \\ & \times (1 + e^{-20(4(R_{A,\text{cov}} + R_{B,\text{cov}} + 2)/3R_{AB} - 1)})^{-1} \end{aligned} \tag{0.5}$$

h_A^l and $\delta h_{CN'_A}^l$ are both fitted constants. EN_A is the electronegativity of the element of atom A, given in the original `xtb` code.

$$X(EN_A, EN_B) = 1 + k_{EN}\Delta EN_{AB}^2 \tag{0.6}$$

$$k_{EN} = 0.02 \text{ in GFN2} \tag{0.7}$$

$$\Delta EN_{AB}^2 = (EN_A - EN_B)^2 \tag{0.8}$$

$$\Pi(R_{AB}, l, l') = \left(1 + k_{A,l}^{\text{poly}}\left(\frac{R_{AB}}{R_{\text{cov},AB}}\right)^{\frac{1}{2}}\right)\left(1 + k_{B,l'}^{\text{poly}}\left(\frac{R_{AB}}{R_{\text{cov},AB}}\right)^{\frac{1}{2}}\right) \tag{0.9}$$

$R_{\text{cov},AB}$ are the summed covalent radii ($R_{\text{cov},A} + R_{\text{cov},B}$), e.g. $R_{\text{cov},H} = 0.32$, $R_{\text{cov},C} = 0.75$ are given in the original `xtb` code. $k_{A,l}^{\text{poly}}$ and $k_{B,l'}^{\text{poly}}$ are element and shell specific constants.

$$Y(\zeta_l^A, \zeta_{l'}^B) = \left(\frac{2\sqrt{\zeta_l^A \zeta_{l'}^B}}{\zeta_l^A + \zeta_{l'}^B}\right)^{\frac{1}{2}} \tag{0.10}$$

Here, ζ_l^A are the STO exponents of the GFN2-xTB AO basis.
Slater Type Orbitals are defined as such:

$$\chi_{\zeta,n,l,m}(r,\theta,\varphi) = NY_{l,m}(\theta,\varphi)r^{n-1}e^{-\zeta r} \tag{0.11}$$

N is a normalisation constant, Y are spherical harmonic funtions, n, l, m are the quantum numbers for the AO. r, θ, φ are polar 3D coordinates. ζ determines the radial extent of the STO, a large value gives rise to a function that is "tight" around the nucleus and a small value gives a more "diffuse" function. This ζ is the one mentioned in the Y term of E_{EHT} and is a value fitted when constructing the basis set, thus it is given to us.

0.6 Fock Matrix for GFN2-xTB

$$\begin{gathered} F_{\mu\nu}^{GFN2-xTB} = H_{\mu\nu}^{EHT} + F_{\mu\nu}^{IES+IXC} + F_{\mu\nu}^{AES} + F_{\mu\nu}^{AXC} + F_{\mu\nu}^{D4}, \\ \forall \mu \in A, \nu \in B \end{gathered} \tag{0.12}$$

0.6.1 Isotropic Electrostatic and Exchange-correlation contribution

$$F_{\mu\nu}^{IES+IXC} = -\frac{1}{2}S_{\mu\nu}\sum_{C}\sum_{l''}(\gamma_{AC,ll''}+\gamma_{BC,l'l''})q_{C,l''} \\ -\frac{1}{2}S_{\mu\nu}(q_{A,l}^{2}\Gamma_{A,l}+q_{B,l'}^{2}\Gamma_{B,l'}) \tag{0.13}$$

l, l', l'' being the angular momenta of the orbitals μ, ν and each of C's orbitals.

$$\Gamma_{A,l} = K_l^{\Gamma}\Gamma_A \tag{0.14}$$

K_l^{Γ} is a shell specific constant common for all elements and Γ_A is an element specific constant.

$$\gamma_{AB,ll'} = \frac{1}{\sqrt{R_{AB}^2+\eta_{AB,ll'}^{-2}}} \tag{0.15}$$

$$\eta_{AB,ll'} = \frac{1}{2}\left[\eta_A(1+k_A^l)+\eta_B(1+k_B^{l'})\right] \tag{0.16}$$

q_l is a partial Mulliken charge. η_A and η_B are element-specific fit parameters, while k_A^l and $k_B^{l'}$ are element-specific scaling factors for the individual shells ($k_A^l = 0$ when $l = 0$).

$$GAP_A = \sum_{l\in A} q_{A,l} \tag{0.17}$$

$$q_{A,l} = \sum_{l'\in B} P_{ll'}S_{ll'} = GOP_l \tag{0.18}$$

0.6.2 Anisotropic Electrostatic and Exchange-correlation contribution

$$F_{\mu\nu}^{AES}+F_{\mu\nu}^{AXC} = \frac{1}{2}S_{\mu\nu}\left[V_S(\boldsymbol{R}_B)+V_S(\boldsymbol{R}_C)\right] \\ +\frac{1}{2}\boldsymbol{D}_{\mu\nu}^{T}\left[\boldsymbol{V}_D(\boldsymbol{R}_B)+\boldsymbol{V}_D(\boldsymbol{R}_C)\right] \\ +\frac{1}{2}\sum_{\alpha,\beta\in\{x,y,z\}} Q_{\mu\nu}^{\alpha\beta}\left[V_Q^{\alpha\beta}(\boldsymbol{R}_B)+V_Q^{\alpha\beta}(\boldsymbol{R}_C)\right] \tag{0.19}$$

$$\boldsymbol{D}^T_{\mu\nu} = \begin{pmatrix} D^x_{\mu\nu} & D^y_{\mu\nu} & D^z_{\mu\nu} \end{pmatrix} \tag{0.20}$$

$$\tag{0.21}$$

$$\begin{aligned} V_S(\boldsymbol{R}_C) = \sum_A \Bigg\{ & \boldsymbol{R}_C^T \Bigg[f_5(R_{AC}) \boldsymbol{\mu}_A R^2_{AC} - \boldsymbol{R}_{AC} 3 f_5(R_{AC}) (\boldsymbol{\mu}^T_A \boldsymbol{R}^2_{AC}) \\ & - f_3(R_{AC}) q_A \boldsymbol{R}_{AC} \Bigg] - f_5(R_{AC}) \boldsymbol{R}^T_{AC} \boldsymbol{\Theta}_A \boldsymbol{R}_{AC} - f_3(R_{AC}) \boldsymbol{\mu}^T_A \boldsymbol{R}_{AC} \\ & + q_A f_5(R_{AC}) \frac{1}{2} \boldsymbol{R}^2_C \boldsymbol{R}^2_{AC} - \frac{3}{2} q_A f_5(R_{AC}) \sum_{\alpha\beta} \alpha_{AB} \beta_{AB} \alpha_C \beta_C \Bigg\} \\ & + 2 f^{\mu_C}_{XC} \boldsymbol{R}^T_C \boldsymbol{\mu}_C - f^{\Theta_C}_{XC} \boldsymbol{R}^T_C \Big[3\boldsymbol{\Theta}_C - \mathrm{Tr}(\boldsymbol{\Theta}_C) \boldsymbol{I} \Big] \boldsymbol{R}_C \end{aligned} \tag{0.22}$$

QUESTION: Should this not be $R^{2,T}_C$, in line 3, term 1?

$$\begin{aligned} V_D(\boldsymbol{R}_C) = \sum_A \Bigg[& \boldsymbol{R}_{AC} 3 f_5(R_{AC}) (\boldsymbol{\mu}^T_A \boldsymbol{R}_{AC}) - f_5(R_{AC}) \boldsymbol{\mu}_A R^2_{AC} + f_3(R_{AC}) q_A \boldsymbol{R}_{AC} \\ & - q_A f_5(R_{AC}) \boldsymbol{R}_C R^2_{AC} + 3 q_A f_5(R_{AC}) \boldsymbol{R}_{AC} \sum_\alpha \alpha_C \alpha_{AC} \Bigg] \\ & - 2 f^{\mu_C}_{XC} \boldsymbol{\mu}_C - 2 f^{\Theta_C}_{XC} \Big[3\boldsymbol{\Theta}_C - \mathrm{Tr}(\boldsymbol{\Theta}_C) \boldsymbol{I} \Big] \boldsymbol{R}_C \end{aligned} \tag{0.23}$$

$$\begin{aligned} V^{\alpha\beta}_Q(\boldsymbol{R}_C) = & - \sum_A q_A f_5(R_{AC}) \left[\frac{3}{2} \alpha_{AC} \beta_{AC} - \frac{1}{2} R^2_{AB} \right] \\ & - f^{\Theta_C}_{XC} \left[3 \boldsymbol{\Theta}^{\alpha\beta}_C - \delta_{\alpha\beta} \sum_\alpha \boldsymbol{\Theta}^{\alpha\alpha}_C \right] \end{aligned} \tag{0.24}$$

$\boldsymbol{\mu}_A$ is the cumulative atomic dipole moment of atom A and $\boldsymbol{\Theta}_A$ is the corresponding traceless quadrupole moment. Traceless simply means that the sum of the diagonal elements is 0. The curly braces and brackets are used in the same way as normal parenthesis for showing order of operations. q_A is the atomic charge of atom A.

$$\Theta^{\alpha\beta}_A = \frac{3}{2} \theta^{\alpha\beta}_A - \frac{\delta_{\alpha\beta}}{2} \left(\theta^{xx}_A + \theta^{yy}_A + \theta^{zz}_A \right) \tag{0.25}$$

$$\theta^{\alpha\beta}_A = \sum_{l' \in A} \sum_l P_l \left(\alpha_A D^\beta_{ll'} + \beta_A D^\alpha_{ll'} - \alpha_A \beta_A S_{ll'} - Q^{\alpha\beta}_{ll'} \right) \tag{0.26}$$

$$q_A = Z_A - GAP_A \tag{0.27}$$

$$\mu_A^\alpha = \sum_{l' \in A} \sum_{l} P_{l'l} (\alpha_A S_{l'l} - D_{l'l}^\alpha) \tag{0.28}$$

$$D_{ll'}^\alpha = \langle \phi_l | \alpha_i | \phi_{l'} \rangle = \langle \phi_l(\alpha_i) | \alpha_i | \phi_{l'}(\alpha_i) \rangle = \int \alpha_i \phi_l^*(\alpha_i) \phi_{l'}(\alpha_i) d\alpha_i \tag{0.29}$$

$$Q_{ll'}^{\alpha\beta} = \langle \phi_l | \alpha_i \beta_i | \phi_{l'} \rangle = \langle \phi_l(\alpha_i) | \alpha_i \beta_i | \phi_{l'}(\beta_i) \rangle = \int \int \alpha_i \beta_i \phi_l^*(\alpha_i) \phi_{l'}(\beta_i) d\alpha_i d\beta_i \tag{0.30}$$

α and β are Cartesian components labled $(x, y, z)^T$ with atom A being centered in $\boldsymbol{R}_A = (x_i, y_i, z_i)^T$ where i is a form of pointer/label dereferencing. $\delta_{\alpha\beta}$ is just the delta function, i.e is is 1 if α and β are the same label and 0 otherwise, this serves to include the term only for the diagonal.

$$\boldsymbol{\Theta}_A = \begin{pmatrix} \Theta_A^{xx} & \Theta_A^{xy} & \Theta_A^{xz} \\ \Theta_A^{yx} & \Theta_A^{yy} & \Theta_A^{yz} \\ \Theta_A^{zx} & \Theta_A^{zy} & \Theta_A^{zz} \end{pmatrix} \tag{0.31}$$

$$\boldsymbol{\mu}_A = \begin{pmatrix} \mu_A^x & \mu_A^y & \mu_A^z \end{pmatrix}^T \tag{0.32}$$

$$\boldsymbol{R}_{AB} = \boldsymbol{R}_A - \boldsymbol{R}_B \tag{0.33}$$

$$R_{AB} = \sqrt{(\boldsymbol{R}_{AB}^x)^2 + (\boldsymbol{R}_{AB}^y)^2 + (\boldsymbol{R}_{AB}^z)^2} \tag{0.34}$$

$$f_n(R_{AB}) = \frac{f_{damp}(a_n, R_{AB})}{R_{AB}^n} = \frac{1}{R_{AB}^n} \frac{1}{1 + 6\left(\frac{R_0^{AB}}{R_{AB}}\right)^{a_n}} \tag{0.35}$$

$$R_0^{AB} = 0.5(R_0^{A'} + R_0^{B'}) \tag{0.36}$$

$$R_0^{A'} = \begin{cases} R_0^A + \frac{R_{max} - R_0^A}{1 + exp[-4(CN_A' - N_{val} - \Delta_{val})]} & \text{if } N_{val} \text{ is given} \\ 5.0 \text{ bohrs} & \text{otherwise} \end{cases} \tag{0.37}$$

$$R_{max} = 5.0 \text{ bohrs} \tag{0.38}$$

$$\Delta_{val} = 1.2 \tag{0.39}$$

R_0^A is a fitted value for 12 elements and 5.0 for the rest. a_n are adjusted global parameters. Where $f_{XC}^{\mu_A}$ and $f_{XC}^{\Theta_A}$ are fitted values.

0.6.3 Dispersion contribution

$$F_{\mu\nu}^{D4} = -\frac{1}{2} S_{\mu\nu}(d_A + d_B), \forall \mu \in A, \nu \in B \tag{0.40}$$

$$\begin{aligned} d_A = \sum_r^{N_{A,ref}} \frac{\partial \xi_A^r(q_A, q_{A,r})}{\partial q_A} \sum_B \sum_s^{N_{B,ref}} \sum_{n=6,8} \\ W_A^r(CN_{cov}^A, CN_{cov}^{A,r}) W_B^s(CN_{cov}^B, CN_{cov}^{B,s}) \xi_B^s(q_B, q_{B,s}) \times \\ s_n \frac{C_n^{AB,ref}}{R_{AB}^n} f_n^{damp,BJ}(R_{AB}) \end{aligned} \tag{0.41}$$

The dispersion coefficient for two reference atoms $C_n^{AB,\text{ref}}$ is evaluated at the reference points, i.e., for $q_A = q_r$, $q_B = q_s$, $CN_{\text{cov}}^A = CN_{\text{cov}}^r$, and $CN_{\text{cov}}^B = CN_{\text{cov}}^s$.

The Gaussian weighting for each reference system is given by:

$$W_A^r(CN_{cov}^A, CN_{cov}^{A,r}) = \sum_{j=1}^{N_{gauss}} \frac{1}{\mathcal{N}} \exp\left[-6j \cdot (CN_{cov}^A - CN_{cov}^{A,r})^2\right] \tag{0.42}$$

with

$$\sum_r^{N_{A,ref}} W_A^r(CN_{cov}^A, CN_{cov}^{A,r}) = 1 \tag{0.43}$$

$\mathcal{N}$ is a normalization constant.

$$\mathcal{N} = \sum_{A,\text{ref}=1}^{N^{A,\text{ref}}} \exp\left[-6j \cdot (CN^A - CN^{A,\text{ref}})^2\right] \tag{0.44}$$

// Write r or ref? CN with or without cov?

The number of Gaussian function per reference system N_{gauss} is mostly one, but equal to three for $CN_{cov}^{A,r} = 0$ and reference systems with similar coordination number.

C_6^{AB} is the pairwise dipole-dipole dispersion coefficients calculated by numerical integration via the Casimir-Polder relation.

$$C_6^{AB} = \frac{3}{\pi}\sum_j w_j \overline{\alpha}_A(i\omega_j, q_A, CN_{cov}^A)\overline{\alpha}_B(i\omega_j, q_B, CN_{cov}^B) \tag{0.45}$$

w_j are the integration weights, which are derived from a trapeziodal partitioning between the grid points $j \in \{2, \ldots, 22\}$.

The isotropically averaged, dynamic dipole-dipole polarizabilites $\overline{\alpha}$ at the jth imaginary frequency $i\omega_j$ are obtained from the self-consistent D4 model; i.e., they are depending on the covalent coordination number and are also charge dependent.

$$\overline{\alpha}_A(i\omega_j, q_A, CN_{cov}^A) = \sum_r^{N_{A,ref}} \xi_A^r(q_A, q_{A,r})\overline{\alpha}_{A,r}(i\omega_j, q_{A,r}, CN_{cov}^{A,r})W_A^r(CN_{cov}^A, CN_{cov}^{A,r}) \tag{0.46}$$

$$\overline{\alpha}_{A,r}(i\omega_j, q_{A,r}, CN_{cov}^{A,r}) = \sum_{A,\text{ref}=1}^{N^{A,\text{ref}}} \alpha^{A,\text{ref}}(i\omega, q_A)W_A^r \tag{0.47}$$

The charge-dependent atomic dynamic polarizability for a single reference system of atom A is given by the product of $\alpha^{A,\text{ref}}(i\omega)$ and its scaling function as:

$$\alpha^{A,\text{ref}}(i\omega, q_A) = \alpha^{A,\text{ref}}(i\omega)\xi_A^r(q_A, q_{A,r}) \tag{0.48}$$

$$\alpha^{A,\text{ref}}(i\omega) = \frac{1}{m}\left[\alpha^{AmXn}(i\omega) - \frac{n}{l}\alpha^{X_l}(i\omega)\xi_A^r(q_X, q_{X,r})\right] \tag{0.49}$$

// The effective nuclear charges $z^{X,\text{ref}}$ entering equation 0.49 are constant values determined once for the respective reference system. (Find out how to get them)

The charge-dependency is included via the empirical scaling function ξ^r_A.

$$\xi^r_A(q_A, q_{A,r}) = \exp\left[3\left\{1 - \exp\left[4\eta_A\left(1 - \frac{Z^{eff}_A + q_{A,r}}{Z^{eff}_A + q_A}\right)\right]\right\}\right] \tag{0.50}$$

where η_A is the chemical hardness taken from ref 98.

Z^{eff}_A is the effective nuclear charge of atom A.

C^{AB}_8 is calculated recursively from the lowest order C^{AB}_6 coefficients.

$$C^{AB}_8 = 3C^{AB}_6\sqrt{\mathcal{Q}^A\mathcal{Q}^B} \tag{0.51}$$

$$\mathcal{Q}^A = s_{42}\sqrt{Z^A}\frac{\langle r^4\rangle^A}{\langle r^2\rangle^A} \tag{0.52}$$

$\sqrt{Z^A}$ is the ad hoc nuclear charge dependent factor.

From the original xTB program we can see that s_{42} is 0.5, and Z^A is the atomic number of A.

$$\sqrt{0.5\left(\frac{r^4}{r^2}\sqrt{Z^A}\right)} \tag{0.53}$$

$\langle r^4\rangle$ and $\langle r^2\rangle$ are simple multipole-type expectation values derived from atomic densities which are averaged geometrically to get the pair coefficients.

CN^A_{cov} is the covalent coordination number for atom A.

q is the atomic charge, so q_A is the atomic charge for atom A.

The scaling parameters in the dispersion model are:

$$a1 = 0.52 \quad | \quad a2 = 5.0 \quad | \quad s6 = 1.0 \quad | \quad s8 = 2.7$$

BJ = Becke-Johnson

$$f_n^{damp,BJ}(R_{AB}) = \frac{R_{AB}^n}{R_{AB}^n + (a_1 \times R_{AB}^{crit} + a_2)^6} \tag{0.54}$$

$$R_{AB}^{crit} = \sqrt{\frac{C_8^{AB}}{C_6^{AB}}} \tag{0.55}$$

$$f_9^{damp,zero}(R_{AB}, R_{AC}, R_{BC}) = \left(1 + 6\left(\sqrt{\frac{R_{AB}^{crit} R_{BC}^{crit} R_{CA}^{crit}}{R_{AB} R_{BC} R_{CA}}}\right)^{16}\right)^{-1} \tag{0.56}$$

0.7 Total Energy for GFN2-xTB

$$\begin{aligned} E_{GFN2-xTB} &= E_{rep}^{(0)} + E_{disp}^{(0,1,2)} + E_{EHT}^{(1)} + E_{IES+IXC}^{(2)} + E_{AES+AXC}^{(2)} + E_{IES+IXC}^{(3)} \\ &= E_{rep} + E_{disp}^{D4'} + E_{EHT} + E_{\gamma} + E_{AES} + E_{AXC} + E_{\Gamma}^{GFN2} \end{aligned} \tag{0.57}$$

0.7.1 Repulsion Energy

$$E_{rep} = \frac{1}{2} \sum_{A,B} \frac{Z_A^{eff} Z_B^{eff}}{R_{AB}} e^{-\sqrt{a_A a_B}(R_{AB})^{(k_f)}} \tag{0.58}$$

$$k_f = \begin{cases} 1 & if A, B \in \{\text{H}, \text{He}\} \\ \frac{3}{2} & otherwise \end{cases} \tag{0.59}$$

Z^{eff} and a are variables fitted for each element. A,B are the labels of atoms. Since we only have C and H in our systems we can simplify this quite a bit in code. R_{AB} is the distance between the A and B atoms.

0.7.2 Extended Hückel Theory Energy

$$E_{EHT} = \sum_{\mu\nu} P_{\mu\nu} H_{\mu\nu}^{EHT} \tag{0.60}$$

$$P_{\mu\nu} = P_{\mu\nu}^0 + \delta P_{\mu\nu} \tag{0.61}$$

$$P^0 = \sum_A P_A^0 \tag{0.62}$$

$$\delta P_{\mu\nu} = ?? \quad \text{comes from the iteration, can be skipped for now} \tag{0.63}$$

Where P_A^0 is the neutral atomic reference density of A. This is known as Superposition of Atomic Densities or SAD.

0.7.3 Isotropic electrostatic and Exchange-correlation energy

Second order

$$E_\gamma = \frac{1}{2} \sum_{A,B}^{N_{atoms}} \sum_{l \in A} \sum_{l' \in B} q_{A,l} q_{B,l'} \gamma_{AB,ll'} \tag{0.64}$$

Third order

$$E_\Gamma^{GFN2} = \frac{1}{3} \sum_{A}^{N_{atoms}} \sum_{l \in A} (q_{A,l})^3 \Gamma_{A,l} \tag{0.65}$$

0.7.4 Anisotropic electrostatic energy

$$
\begin{aligned}
E_{AES} &= E_{q\mu} + E_{q\Theta} + E_{\mu\mu} \\
&= \frac{1}{2}\sum_{A,B}\{f_3(R_{AB})[q_A(\boldsymbol{\mu}_B^T\boldsymbol{R}_{BA}) + q_B(\boldsymbol{\mu}_A^T\boldsymbol{R}_{AB})] \\
&\quad + f_5(R_{AB})[q_A\boldsymbol{R}_{AB}^T\boldsymbol{\Theta}_B\boldsymbol{R}_{AB} + q_B\boldsymbol{R}_{AB}^T\boldsymbol{\Theta}_A\boldsymbol{R}_{AB} \\
&\quad - 3(\boldsymbol{\mu}_A^T\boldsymbol{R}_{AB})(\boldsymbol{\mu}_B^T\boldsymbol{R}_{AB}) + (\boldsymbol{\mu}_A^T\boldsymbol{\mu}_B)R_{AB}^2]\}
\end{aligned} \tag{0.66}
$$

0.7.5 Anisotropic XC energy

$$
E_{AXC} = \sum_A (f_{XC}^{\mu_A}|\boldsymbol{\mu}_A|^2 + f_{XC}^{\Theta_A}||\boldsymbol{\Theta}_A||^2) \tag{0.67}
$$

What norms are these?

0.7.6 Dispersion Energy

$$E_{disp}^{D4'} = -\sum_{A>B}\sum_{n=6,8} s_n \frac{C_n^{AB}(q_A, CN_{cov}^A, q_B, CN_{cov}^B)}{R_{AB}^n} f_{damp,BJ}^{(n)}(R_{AB})$$
$$- s_9 \sum_{A>B>C} \frac{(3cos(\theta_{ABC})cos(\theta_{BCA})cos(\theta_{CAB})+1)C_9^{ABC}(CN_{cov}^A, CN_{cov}^B, CN_{cov}^C)}{(R_{AB}R_{AC}R_{BC})^3} \tag{0.68}$$
$$\times f_{damp,zero}^{(9)}(R_{AB}, R_{AC}, R_{BC}).$$

The term in the second line is the three-body Axilrod– Teller–Muto (ATM) (What is this??????) term and the last line is the corresponding zero-damping function for this term.

The damping and scaling parameters in the dispersion model are:

$$s6 = 1.0 \quad | \quad s8 = 2.7 \quad | \quad s9 = 5.0$$

C_9^{ABC} is the triple-dipole constant[2]:

$$C_9^{ABC} = \frac{3}{\pi}\int_0^\infty \alpha^A(i\omega)\alpha^B(i\omega)\alpha^C(i\omega)d\omega \tag{0.69}$$

The three-body contribution is typically $< 5-10\%$ of E_{disp}, so it is small enough that we can reasonably approximate the coefficients by a geometric mean as[2]:

$$C_9^{ABC} \approx -\sqrt{C_6^{AB}C_6^{AC}C_6^{BC}} \tag{0.70}$$

θ_{ABC} is the angle between the two edges going from B to the other two atoms. θ_{BCA} is the angle between the edges going from C to the other two and so on.

[2]https://www.researchgate.net/publication/43347348_A_Consistent_and_Accurate_Ab_Initio_Parametrization_of_Density_Func D_for_the_94_Elements_H-Pu

0.7.7 SAD - Superposition of Atomic Densities

The superposition of atomic densities(SAD) is an approach to obtain a good approximation of a collection of atoms, to be used as an initial guess for solving the self-consistent field(SCF) equation.

As originally implemented in DISCO, the molecular electron density can be obtained by adding the densities of all the constituting atoms.

This is how we get the density matrix for an isolated atom? equation 15 from: (https://sci-hub.box/10.1002/jcc.540030314)

$$D_{ij} = \sum_{a}^{occ} c_{ia} c_{ja} \tag{0.71}$$

To get the coefficients we need to solve SCF for each atom? this is supposedly cheap, but idk how to do it. (https://sci-hub.box/10.1002/jcc.20393) Though the math for Direct SCF Approach is given in this paper at equation 10: (https://sci-hub.box/10.1002/jcc.540030314). This is probably how.

The SAD method is then the sum of all of these?

Equation 2 in the GFN2 paper talks about "superposition of (neutral) atomic reference densities". Is this relevant?

Direct SCF Approach

$$\begin{aligned}
\Delta F_{ab} =& (c_{ia}c_{jb} + c_{ja}c_{ib}) \\
&\Delta F_{ij} + (c_{ia}c_{kb} + c_{ka}c_{ib}) \\
&\Delta F_{ik} + (c_{ia}c_{lb} + c_{la}c_{ib}) \\
&\Delta F_{il} + (c_{ja}c_{kb} + c_{ka}c_{jb}) \\
&\Delta F_{jk} + (c_{ja}c_{lb} + c_{la}c_{jb}) \\
&\Delta F_{jl} + (c_{ka}c_{lb} + c_{la}c_{kb})\Delta F_{kl} \\
&= l_{ijkl}(4E_{ij}^{ab}D_{kl} + 4D_{ij}E_{kl}^{ab} - E_{ik}^{ab}D_{jl} - D_{ik}E_{jl}^{ab} - E_{il}^{ab}D_{jk} - D_{il}E_{jk}^{ab})
\end{aligned} \tag{0.72}$$

where

$$E_{ij}^{ab} = c_{ia}c_{jb} + c_{ja}c_{ib} \tag{0.73}$$

Equation 18 from (https://sci-hub.box/https://doi.org/10.1021/acs.chemrev.5b00584) uses ρ_0 which is the superposition of neutral atom densities:

$$\rho_0 = \sum_A \rho_0^A \tag{0.74}$$

0.8 AI Declaration

look at what needs to be in the AI section somewhere on KU's website.

Part II

Appendicies

An appendix

A

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Bibliography

Part III

Articles