Bachelor's thesis
Bachelor's Program in Physical Sciences
Physics (Broad)

Mapping the electrostatic potential of a molecule from its electron density using parallel computing

Nico Timothy Toikka

13.9.2021

Supervisors: Dr. Ondřej Krejčí, Dr. Filippo Federici Canova

Examiner: Docent Antti Kuronen

Helsingin Yliopisto
Faculty of Science

PL 64 (Gustaf Hällströmin katu 2a)
00014 Helsingin yliopisto

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Degree programme
Faculty of Science	Bachelor's Program in Physical Sciences Physics (Broad)

Tekijä — Författare — Author

Nico Timothy Toikka

Työn nimi — Arbetets titel — Title

Mapping the electrostatic potential of a molecule from its electron density using parallel computing

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidantal — Number of pages
Bachelor's thesis	September 13, 2021	48

Tiivistelmä — Referat — Abstract

Aalto University's Surfaces and Interfaces at the Nanoscale (SIN) group are developing a machine learning based method to recover the electrostatic field around a molecule from Atomic Force Microscopy (AFM) measurements. The model, Electrostatic Discovery Atomic Force Microscopy, uses reference simulations to support the characterization of the electrostatic properties. This thesis offers a GPU implemented parallel solution to calculate the electrostatic potential for possible use in the model. By first recovering the electron density of the system, the electrostatic potential can be calculated by solving Poissons' equation via Fourier transform. Basics of electron density calculations, fast Fourier transform (FFT) and parallel computing are discussed. To increase the accuracy of Fourier transforms in the computations, a convolution and a Gaussian density based spreading methods for nuclear charges are presented. Results using these methods give feasible maps of the electrostatic potential, with overall correct behaviour at the points of the nuclei and while moving away from them, but reference images or values are required for a complete assessment of the results. Results and analysis of calculations for a bromodicholorobenzene molecule are provided, with additional images for a dihydrogen molecule. The GPU implementation using Nvidia's CUDA model is shown to be approximately 35 times faster than a CPU counterpart implemented using Python and Numpy's FFT library.

Avainsanat — Nyckelord — Keywords

Molecule's electrostatic potential, Electron density, Parallel computing, Fast Fourier transform

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Degree programme
Matemaattisluonnontieteellinen tiedekunta	Fysikaalisten tieteiden kandiohjelma Fysiikka, laaja-alainen

Tekijä — Författare — Author

Nico Timothy Toikka

Työn nimi — Arbetets titel — Title

Molekyylin sähköisen potentiaalin kartoittaminen elektronitiheydestä rinnakkaislaskennan avulla

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidantal — Number of pages
Kandidaatintutkielma	7.9.2021	48

Tiivistelmä — Referat — Abstract

Aalto yliopiston ryhmä Surfaces and Interfaces at the Nanoscale (SIN) on kehittämässä koneoppimiseen pohjautuvaa menetelmää selvittääkseen molekyylin staattisen sähkökentän atomivoimamikroskopia-mittausten perusteella. Tässä menetelmässä, Electrostatic Discovery Atomic Force Microscopy, ryhmä käyttää viitesimulaatioita sähköisten ominaisuuksien luonnehtimiseen. Tämä tutkielma esittää keinon laskea molekyylin sähköisen potentiaalin menetelmässä käytettyä luonnehtimista varten. Lähtien liikkeelle molekyylin elektronitiheydestä sähköinen potentiaali voidaan ratkaista Poissonin yhtälöstä Fourier-muunnoksen avulla. Laskenta toteutaan Nvidian CUDA-rinnakkaislaskentamallilla. Tutkielma kertoo elektronitiheyslaskennan, nopean Fourier-muunnoksen ja rinnakkaislaskennan perusteet sekä Poissonin yhtälön ratkaisun Fourier-muunnoksella. Fourier-muunnoksen tiedetään tuottavan erheellisiä tuloksia nopeasti muuttuville arvoille, minkä takia tutkielma esittää myös kaksi keinoa molekyylin ydinten varausten levittämiselle. Näillä menetelmillä tuotetut sähköisen potentiaalin kartat pitävät sisällään sähköisen potentiaalin perusominaisuuksia, mutta ilman verrattavia tuloksia karttojen tarkkuutta ei voida vahvistaa. Rinnakkaislaskettujen simulaatioiden osoitetaan toimivan noin 35 kertaa nopeammin kuin Pythonin Numpy-kirjastolla tuotetut sarjassa lasketut simulaatiot. Tutkielma pitää sisällään tulokset ja analyysin bromidiklooribentseenimolekyylille sekä vastaavat kuvaajat divetymolekyylille.

Avainsanat — Nyckelord — Keywords

Molekyylin sähköinen potentiaali, Elektronitiheys, Rinnakkaislaskenta, Nopea Fourier-muunnos

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisällys

Symboliluettelo

ϵ_0	Vacuum permittivity
$\hat{H}$	Hamilton operator
$\hat{T}$	Kinetic energy operator in the simplified Hamilton operator
$\hat{T}_{cc}$	Cluster operator in the coupled clusters method
$\hat{V}_{ee}$	Electron-electron operator in the simplified Hamilton operator
$\hat{V}_{ext}$	External potential operator in the simplified Hamilton operator
∇^2	Laplacian operator
$\mathcal{F}[f]$	Fourier transform of function f
$\mathcal{F}^{-1}[f]$	Inverse Fourier transform of function f
Φ_{SD}	Slater determinant
Ψ_i	Wave function of state i
ρ	Charge or electron density
σ	Variance
$\mathbf{P}_{\mu\nu}$	Density matrix
$\widetilde{f}$	Function f in frequency domain
k	Frequency domain variable in the Fourier transformation
M_A	Mass of nucleus A in a molecule
Q	Charge
R_A	Position of nucleus A in a molecule
V	Electrostatic potential

1. Introduction

In their work on the Electrostatic Discovery Atomic Force Microscopy Oinonen et al. [1] aim to create a machine learning based method that provides "immediate quantitative maps of the electrostatic potential directly from Atomic Force Microscopy (AFM) images with functionalized tips." The method is based upon comparing the electrostatic field of a molecular system to the AFM image of the system through an artificial neural network. Reference simulations are used to support the characterization of the electrostatic properties of a molecular system. This thesis aims to offer a method for enhancing the precision and the performance of the reference simulations.

Starting points for creating the electrostatic maps are the electron densities of the systems. We will look at the basics of how these densities are calculated and what are the approximations and methods used in the calculations. From the electron densities we are able to recover the electrostatic potentials with the use of Poisson's equation and it's Fourier transformation solution, which can be implemented via the fast Fourier transformation (FFT). Due to the sharp spikes in charge density caused by the nuclei of the system we also present two methods to spread the charge to the space near the nuclei. At the end of the thesis results for a bromodichlorobenzene molecule will be presented and analyzed, with additional images of a dihydrogen system provided in Appendix A.

For fast results the spreading of the nuclear charge and the solution for Poisson's equation are both implemented using Nvidia's CUDA model for GPU computing. For this, a quick introduction to parallel computing and the CUDA model is given. We will also compare the performance of the GPU model to a serial implementation of the same calculations using Python and Numpy's FFT library.

2. Theory

2.1 The Schrödinger equation

A necessary tool for analyzing any quantum system, such as atoms or molecules, is the time-independent, non-relativistic *Schrödinger equation* [2]

$$\hat{H}\Psi_i(\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_N, \mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_M) = E_i\Psi_i(\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_N, \mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_M) \tag{2.1}$$

where N is the number of electrons, M is the number of nuclei in the system and $\hat{H}$ is the Hamilton operator for the system in atomic units defined as

$$\hat{H} = -\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2 - \frac{1}{2}\sum_{A=1}^{M}\frac{1}{M_A}\nabla_A^2 - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>1}^{N}\frac{1}{r_{ij}} + \sum_{A=1}^{M}\sum_{B>1}^{M}\frac{Z_A Z_B}{R_{AB}} \tag{2.2}$$

The first two terms of the Hamilton operator describe the kinetic energy of the electrons and nuclei respectively and use the Laplacian operator, which has a Cartesian coordinate representation:

$$\nabla_k^2 = \frac{\partial^2}{\partial x_k^2} + \frac{\partial^2}{\partial y_k^2} + \frac{\partial^2}{\partial z_k^2} \tag{2.3}$$

The second term in the Hamiltonian operator includes the mass of a nucleus A, M_A, in multiples of the mass of electron, as is common when using atomic units. The last three terms in the operator represent the electrostatic interaction between the electrons and the nuclei, the electrons interaction between each other and the nucleus' interaction between each other respectively. Other variables in the Hamilton operator are the distance between particle p and q, r_{pq}, and the nuclear charge of a nucleus A, Z_A.

The Schrödinger equation also includes the *wave function* of the state i, Ψ_i, and the numerical value for the energy of the same state, E_i. The wave function is dependent on the position of the electrons, $\mathbf{r}_i$, and their spin coordinates, s_i. These variables are combined together to the variable $\mathbf{x}_i$. The wave function also depends on the position of the nuclei, $\mathbf{R}_i$. All the necessary information used to analyze the

system is thus included in the wave function, and solving the wave function is the main goal of most quantum chemical and physical approaches [2].

To simplify solving the Shrödinger equation Max Born and J. Robert Oppenheimer proposed an approximation in their 1927 paper "Zur Quantentheorie der Molekeln" [3]. In the now well known *Born-Oppenheimer approximation* the nuclei are thought as fixed in space while the electrons move in a field around them. Basis for this approximation is in the much higher mass of the nuclei compared to electrons mass, which results in the nuclei moving slower than the electrons. Since the nuclei are now thought as fixed in space, their kinetic energy reduces to zero and leads to a simplified Hamilton operator [2]

$$\hat{H}_{elec} = -\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>1}^{N}\frac{1}{r_{ij}} = \hat{T} + \hat{V}_{ext} + \hat{V}_{ee} \tag{2.4}$$

The last term from equation 2.2 is also dropped, as it becomes a constant due to the distance between the nuclei, R_{AB}, staying the same, and so we will move from looking at the complete system of nuclei and electrons to looking only at the electrons of the system. This in turn leads to a new Schrödinger equation

$$\hat{H}_{elec}\Psi_{elec} = E_{elec}\Psi_{elec} \tag{2.5}$$

where E_{elec} is a part of the equation for total energy

$$E_{tot} = E_{elec} + E_{nuclei} \quad \text{with} \quad E_{nuclei} = \sum_{A=1}^{M}\sum_{B>1}^{M}\frac{Z_A Z_B}{R_{AB}} \tag{2.6}$$

From now on we will drop the subscript *elec* and only consider the electronic problem equations 2.4 and 2.5.

2.1.1 Properties of the wave function

The wave function has been proven to be unobservable, but it also has been shown that a physical interpretation of the wave function can be found from its square

$$|\Psi(\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_N)|^2 \, \mathrm{d}\mathbf{x}_1 \, \mathrm{d}\mathbf{x}_2 \ldots \mathrm{d}\mathbf{x}_N \tag{2.7}$$

which tells us the probability of finding $1, 2, ..., N$ electrons in volume elements $\mathrm{d}\mathbf{x}_i$ [2]. It should be noted, that since electrons cannot be distinguished from each other, changing their coordinates doesn't change the value given by equation 2.7. This follows from fermions', which include electrons, antisymmetric wave function property

$$\Psi(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_i, \mathbf{x}_j, ..., \mathbf{x}_N) = -\Psi(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_j, \mathbf{x}_i, ..., \mathbf{x}_N) \tag{2.8}$$

Looking at a space Ω with N electrons it is easily understood that the probability for the space to include all these electrons is exactly 1. Mathematically this is expressed as an integral

$$\int \cdots \int |\Psi(\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_N)|^2 \, \mathrm{d}\mathbf{x}_1 \, \mathrm{d}\mathbf{x}_2 \, ... \, \mathrm{d}\mathbf{x}_N = 1 \tag{2.9}$$

When this equation is fulfilled, the wave function is considered normalized. It is standard to only consider normalized wave functions.

2.1.2 Finding a solution for the wave function

Despite the previous simplifications and our understanding of the wave function a clear solution is generally unachievable, disregarding some exceptions such as the hydrogen atom [2]. More approximations and methods are needed to find appropriate solutions. We will now introduce the basics of the *Hartree-Fock approximation*, which is used in our method for calculating the electron densities, coupled clusters method. Both of these methods are too extensive to be precisely looked at in this thesis.

The Hartree-Fock approximation uses N physically appropriate one-electron wave functions $\chi_i(\mathbf{x}_i)$ (*spin orbitals*), so we can approximate the N-electron wave function as their antisymmetrized product.

$$\Psi_0 \approx \Phi_{SD} = \frac{1}{\sqrt{N!}} \begin{vmatrix} \chi_1(\mathbf{x}_1) & \chi_2(\mathbf{x}_1) & \cdots & \chi_N(\mathbf{x}_1) \\ \chi_1(\mathbf{x}_2) & \chi_2(\mathbf{x}_2) & \cdots & \chi_N(\mathbf{x}_2) \\ \vdots & \vdots & & \vdots \\ \chi_1(\mathbf{x}_N) & \chi_2(\mathbf{x}_N) & \cdots & \chi_N(\mathbf{x}_N) \end{vmatrix} \tag{2.10}$$

$$= \frac{1}{\sqrt{N!}} \det\{\chi_1(\mathbf{x}_1), \chi_2(\mathbf{x}_2), \ldots, \chi_N(\mathbf{x}_N)\} \tag{2.11}$$

This is known as the *Slater determinant* and it is used to approximate the ground-state wave function Ψ_0.

As mentioned before we use a *post-Hartree-Fock method* the coupled clusters method in our computation of the electron densities. In the article "Coupled Cluster Theory in Materials Science" [4] the authors describe the coupled cluster theory to "be based upon a systematically improvable many-electron wavefunction ansatz and derived without using uncontrolled approximations". The theory uses an exponential ansatz to solve the wave function

$$\Psi = \exp\left(\hat{T}_{cc}\right)\Phi_{SD} \tag{2.12}$$

where $\hat{T}_{cc}$ is the cluster operator which consists of a sum of excitation operators T_i. Excitation operator itself contains multiple Slater determinants multiplied by their

corresponding amplitudes, which leads to increased accuracy. Due to the complexity of the method, we will not have a further look at it, but for studying the theory Zhang and Grüneis suggest the literature [5], [6], [7] and [8].

2.2 Electron density and basis sets

We will now introduce a new variable called the *electron density*, which comes rather intuitively from the probability interpretation of the wave function [2]. It is defined as

$$\rho(\mathbf{r}_1) = N \int \cdots \int |\Psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)|^2 \, \mathrm{d}s_1 \, \mathrm{d}\mathbf{x}_2 \ldots \mathrm{d}\mathbf{x}_N \tag{2.13}$$

and it tells us the probability of finding any of the N electrons within the volume element $\mathrm{d}\mathbf{r}_1$ with arbitrary spin. From this follows that by integrating the electron density we will receive the number of electrons (compare to equations 2.13 and 2.9). As with the wave functions probability interpretation, the electron density variable can be observed with for example X-ray diffraction.

One notable property of the electron density is that it always has a maximum with a finite value at a position of an atom, as the electrons are drawn towards the positively charged nuclei. This is also seen as a discontinuity and a cusp in the gradient of the electron density. The properties of the cusp have been recognized to relate to the nuclear charge, Z, of nucleus A according to equation

$$\lim_{r \to r_A} \left[\frac{\partial}{\partial r} + 2Z_A \right] \bar{\rho}(\mathbf{r}) = 0 \tag{2.14}$$

where $\bar{\rho}(\mathbf{r})$ is the spherical average of $\rho(\mathbf{r})$. Electron density also decays exponentially for large distances from all nuclei of a single molecule

$$\rho(\mathbf{r}) \propto \exp\left(-2\sqrt{2I}|\mathbf{r}|\right) \tag{2.15}$$

where I is the first ionization energy of the system [2].

From these properties and the fact that we can recover the number of electrons in the system as was described before, we can obtain the amount of electrons, the position of the nuclei and their charges. This information is all that is required to construct the Hamilton operator and to determine all of the molecular properties of the system. Although, even with all the necessary information about the system, solving the Schrödinger equation can prove to be difficult.

The usefulness of the electron density as a variable has been shown by Hohenberg and Kohn [9] with their first and second theorem. The first theorem states that "the external potential $V_{ext}(\mathbf{r})$ is (to within a constant) a unique functional of $\rho(\mathbf{r})$; since, in turn $V_{ext}(\mathbf{r})$ fixes $\hat{H}$ we see that the full many particle ground state is a unique

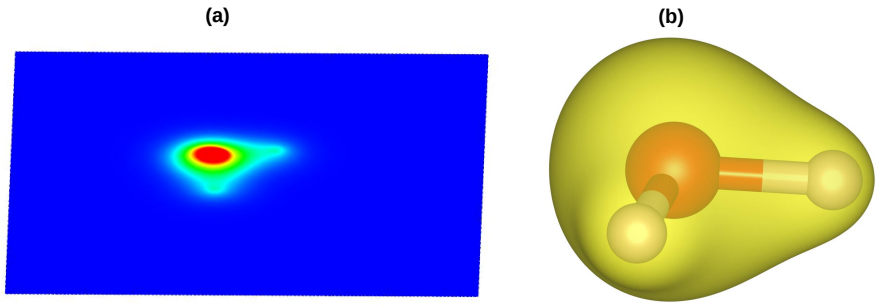


Figure 2.1: Electron density of a water molecule from coupled clusters method: **(a)** mapped onto a plane and **(b)** as an envelope around the molecule at a constant $1.5 \cdot 10^{-5} \frac{1}{\text{Å}^3}$ density. Adapted from [2] Figure 2.1.

.

functional of $\rho(\mathbf{r})$," and the second theorem is summarized by Koch and Holthausen [2] as "this theorem [the second Hohenberg-Kohn theorem] states that $F_{HK}[\rho]$, the functional that delivers the ground state energy of the system, delivers the lowest energy if and only if the input density is the true ground state density, ρ_0." These *Hohenberg-Kohn theorems* are described by Koch and Holthausen as representing "the major theoretical pillar on which all modern day density functional theories [methods for electronic structure calculations] are erected." The theorems are especially relevant in *density functional theory* for calculating electronic structure, although that theory is not used in this thesis. From a computational point of view the electron density is preferable variable to the wave function, as it contains less variables and in turn requires less memory.

2.2.1 Electron density matrices

For us to use the electron densities in computations a more discrete representation than the integral in equation 2.13 is required. So we introduce *linear-combination-of-atomic-orbitals* (LCAO) ansatz [2], which uses a set of size L predefined basis functions, η_μ (basis set). These basis functions are atomic orbitals centered on atoms, $\mathbf{R}_\mu$, of the calculated system. The $c_{\mu i}$ are coefficients that are varied while the Schrödinger

equation is solved.

$$\varphi_i(\mathbf{r}) = \sum_{\mu=1}^{L} c_{\mu i}\eta_\mu|\mathbf{r} - \mathbf{R}_\mu| \tag{2.16}$$

If the set of basis functions is complete, meaning $L = \infty$, we would have an exact representation of the orbitals. This clearly isn't the case, as we can only have a finite number of functions given in real life.

If we know the probability of an electron being at position $\mathbf{r}$ for all the orbitals, we can add them together, which would then result in the electron density at that position

$$\rho(\mathbf{r}) = \sum_{i=1}^{N} |\varphi_i(\mathbf{r})|^2. \tag{2.17}$$

We can substitute equation 2.16 to this equation and get

$$\rho(\mathbf{r}) = \sum_{i=1}^{N}\sum_{\mu=1}^{L}\sum_{\nu=1}^{L} c_{\nu i}c_{\mu i}\eta_\nu\eta_\mu = \sum_{\mu=1}^{L}\sum_{\nu=1}^{L} \mathbf{P}_{\mu\nu}\eta_\nu\eta_\mu \tag{2.18}$$

where the sum over N has been combined to *density matrix* $\mathbf{P}_{\mu\nu}$. When the basis set is known, the density matrix allows to recover all the necessary information about the computed system.

2.2.2 Basis sets

In their book Koch and Holthausen [2] describe three types of basis sets that are used with the LCAO: *Slater-type-orbitals* (STO), *Gaussian-type-orbitals* (GTO) and *contracted Gaussian functions* (CGF). There are other basis sets that use for example a numerical or a plane wave approach, but these sets require their own calculations and assumptions instead of the ones used here.

As was mentioned when discussing the Hartree-Fock approximation, one of the used approximations is to use single electron wave functions. This kind of behaviour is emphasized in Slater-type orbitals which mimic the eigenfunctions of the hydrogen atom.

$$\mu^{\mathrm{STO}} = Nr^{n-1}\exp(-\zeta r)Y_{lm}(\Theta\phi) \tag{2.19}$$

with N being the normalization factor used to ensure property $\langle\mu_i|\mu_i\rangle = 1$, n is the principal quantum number, ζ the orbital exponent and Y_{lm} the spherical harmonics used to describe the angular part of the function. The orbital exponent represents how compact or diffuse the basis function is, with larger values meaning a more compact function. The integers l and m are the quantum numbers of the orbital. STO exhibit

physically correct properties when approaching or moving away from the nuclei, while the later described Gaussian based orbitals fall of too rapidly with the increase of distance and have a slope of zero instead of a cusp in it's gradient when approaching the nuclei. STO's usability is limited by their complexity when integrating them, which has led to their decreased use.

GTOs have the general form

$$\mu^{\mathrm{GTO}} = N x^l y^m z^n \exp\bigl(-\alpha r^2\bigr) \tag{2.20}$$

where N is a normalization factor and α is the orbital exponent. Integers l, m, n and their sum $L = l + m + n$ are used to classify the GTO as s-functions ($L = 0$), p-functions($L = 1$), d-functions ($L = 2$) and so on (following the naming convention of atomic orbitals). GTOs are preferred due to the computational advantages coming from highly developed algorithms for them.

To have a better accuracy than a single GTO, GTOs can be linearly combined with contraction coefficients $d_{a\tau}$, giving one contracted Gaussian functions.

$$\mu_\tau^{\mathrm{CGF}} = \sum_a^A d_{a\tau} \mu_a^{\mathrm{GTO}} \tag{2.21}$$

Contraction coefficients are often chosen to help the CGFs functions resemble STO functions more closely.

In this project we choose to use the Gaussian function based basis set cc-PVDZ (correlation-consistent polarized valence-only double zeta) in our calculations. A description and a study for this and other GTOs can be found from the work of Thom Dunning Jr. and his coworkers, such as in the article "Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen" [10], which is a part of a larger series of study of the GTOs.

2.3 Fast Fourier transform

Any smooth – especially periodic – function can be decomposed to and represented with the frequencies it consists of. This kind of representation is called the *Fourier transform* and it can be calculated as an integral or a series, or the series can be sampled to get the *discrete Fourier transform* [11]. With similar methods an inverse Fourier transform of the frequency representation of a function can be calculated to return the original function. As with basis sets previously, we are limited to looking at the discrete Fourier transformation, since calculating an integral or an infinite sum over real space isn't possible on computers.

Naive computation of the discrete Fourier transform has a complexity of $\mathcal{O}(N^2)$ for a transform with size N. To reduce the complexity many fast Fourier transform (FFT) algorithms have been implemented for different kinds of inputs and systems, with historically most notable being the Cooley-Tukey FFT algorithm [12]. These algorithms have been proven to achieve a complexity of $\mathcal{O}(N\log N)$ greatly reducing the computational complexity. FFT algorithms can also be conveniently implemented in multiple dimensions through the row-column method and they're parallelizable to increase their efficiency for large N.

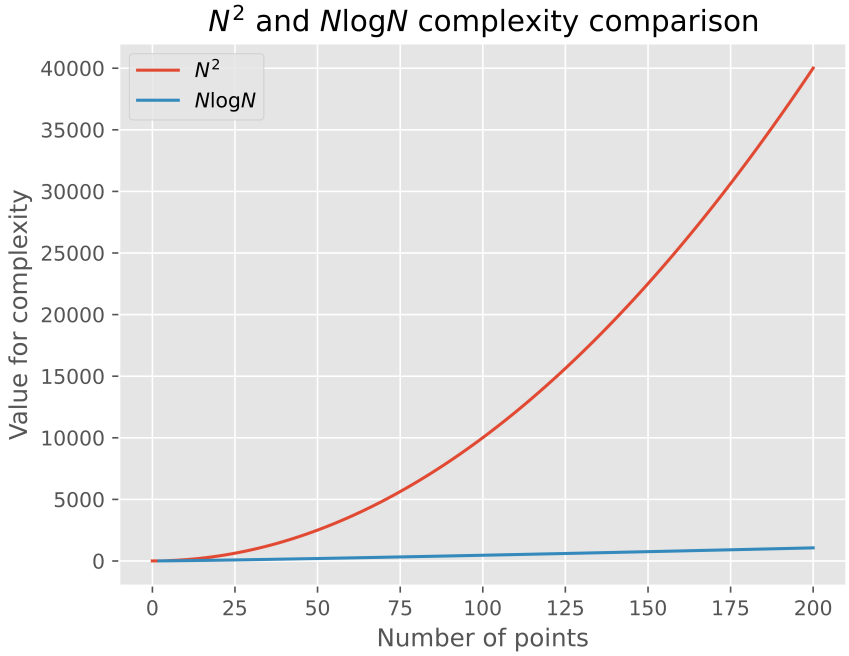


Figure 2.2: Values for N^2 and $N \log N$ plotted against each other.

2.3.1 3D discrete Fourier transform

In CUDA's fast Fourier transfrom library cuFFT [13] the three dimensional discrete Fourier transformation (DFT) for an array $\mathbf{X_n}$ of size $\mathbf{N} = (N_1, N_2, N_3)$, with index $\mathbf{n} = (n_1, n_2, n_3)$ and an integer size N_d for each dimension, from its spatial domain into its frequency domain (*forward transform*) is defined as

$$\mathcal{F}[\mathbf{X_k}] = \widetilde{\mathbf{X}}_\mathbf{k} = \sum_{\mathbf{n}=0}^{\mathbf{N}-1} \mathbf{X_n} \exp\left(-2\pi i \frac{\mathbf{kn}}{\mathbf{N}}\right) \tag{2.22}$$

with $\frac{\mathbf{n}}{\mathbf{N}} = \left(\frac{n_1}{N_1}, \frac{n_2}{N_2}, \frac{n_3}{N_3}\right)$ and $\sum_{\mathbf{n}=0}^{\mathbf{N}-1} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \sum_{n_3=0}^{N_3-1}$. Similarly to the spatial domain variable $\mathbf{n}$ the frequency domain variable $\mathbf{k} = (k_1, k_2, k_3)$.

The DFT from frequency domain into spatial domain (*inverse transform, iDFT*) in cuFFT is equal to forward transform, but with the sign in the exponent changed to be positive.

$$\mathcal{F}^{-1}\left[\widetilde{\mathbf{X}}_\mathbf{n}\right] = \mathbf{X}_\mathbf{n} = \sum_{\mathbf{k}=0}^{\mathbf{N}-1} \widetilde{\mathbf{X}}_\mathbf{k} \exp\left(2\pi i \frac{\mathbf{kn}}{\mathbf{N}}\right) \tag{2.23}$$

The relationship between the DFT and iDFT requires their exponents to have different signs, but there is also a requirement that the product of the normalization constants of DFT and iDFT is equal to $\frac{1}{N}$, where N is the total number of points in the transform, which generalizes to $\frac{1}{\prod_{i=1}^{3} N_i}$ in three dimensions. These requirements are due to that the inverse transform of a forward transform should be the original array [12] and the requirements are easily verified by substituting $\widetilde{\mathbf{X}}_\mathbf{n}$ from equation 2.22 to equation 2.23 [12]

$$\mathbf{X}_\mathbf{n} = \sum_{\mathbf{n}=0}^{\mathbf{N}-1} \mathbf{X}_\mathbf{n} \sum_{\mathbf{k}=0}^{\mathbf{N}-1} \exp\left(2\pi i \frac{\mathbf{kn}}{\mathbf{N}} - 2\pi i \frac{\mathbf{kn}}{\mathbf{N}}\right) \tag{2.24}$$

$$\mathbf{X}_\mathbf{n} = \sum_{\mathbf{n}=0}^{\mathbf{N}-1} \mathbf{X}_\mathbf{n} = \prod_{i=1}^{3} N_i \cdot \mathbf{X}_\mathbf{n} \tag{2.25}$$

from which we can see that a normalization constant must be added to either the forward or the inverse transform.

Since cuFFT doesn't force any convention regarding the normalization constant, we choose to use normalization constant 1 for the DFT as in equation 2.22 and a normalization constant of $\frac{1}{\prod_{i=1}^{3} N_i}$ for the iDFT. The iDFT will then be written as

$$\mathcal{F}^{-1}\left[\widetilde{\mathbf{X}}_\mathbf{n}\right] = \mathbf{X}_\mathbf{n} = \frac{1}{\prod_{i=1}^{3} N_i} \sum_{\mathbf{k}=0}^{\mathbf{N}-1} \widetilde{\mathbf{X}}_\mathbf{k} \exp\left(2\pi i \frac{\mathbf{kn}}{\mathbf{N}}\right) \tag{2.26}$$

Relevant properties of the DFT

For this thesis a relevant property of the DFT is the linearity of the DFT,

$$\mathcal{F}[af] = a \cdot \mathcal{F}(f) \tag{2.27}$$

for any constant a. Linearity follows from the definition in equation 2.22

Proof:

$$\mathcal{F}[a\mathbf{X}_\mathbf{n}] = \sum_{\mathbf{n}=0}^{\mathbf{N}-1} a \cdot \mathbf{X}_\mathbf{n} \exp\left(-2\pi i \frac{\mathbf{kn}}{\mathbf{N}}\right) \tag{2.28}$$

$$= a \cdot \sum_{\mathbf{n}=0}^{\mathbf{N}-1} \mathbf{X}_\mathbf{n} \exp\left(-2\pi i \frac{\mathbf{kn}}{\mathbf{N}}\right) \tag{2.29}$$

$$= a \cdot \mathcal{F}[\mathbf{X}_\mathbf{n}] \tag{2.30}$$

Another important property for the FFT algorithms is the Hermitian symmetry of the DFT for real-valued input arrays

$$\mathbf{X_n} = \mathbf{X^*_{N-n}} \tag{2.31}$$

where $\mathbf{X}^*$ is the complex conjugate of $\mathbf{X}$. This property is commonly utilized for real-to-complex or complex-to-real transformations, since it can be used to mirror the points in the transformation and thus reduce the required number of calculations [12].

Proof:

$$\mathbf{X^*_{N-n}} = \left(\sum_{\mathbf{k}=0}^{\mathbf{N}-1} \widetilde{\mathbf{X}}_\mathbf{k} \exp\left(2\pi i \frac{\mathbf{k}(\mathbf{N}-\mathbf{n})}{\mathbf{N}}\right)\right)^* \tag{2.32}$$

$$= \left(\sum_{\mathbf{k}=0}^{\mathbf{N}-1} \widetilde{\mathbf{X}}_\mathbf{k} \exp\left(-2\pi i \frac{\mathbf{kn}}{\mathbf{N}}\right)\right)^* \tag{2.33}$$

$$= \sum_{\mathbf{k}=0}^{\mathbf{N}-1} \widetilde{\mathbf{X}}_\mathbf{k} \exp\left(2\pi i \frac{\mathbf{kn}}{\mathbf{N}}\right) \tag{2.34}$$

$$= \mathbf{X_n} \tag{2.35}$$

Both of these properties follow similarly for the iDFT.

2.3.2 Cooley-Tukey FFT algorithm

cuFFT implements multiple algorithms for calculating the DFT of an array, but for arrays which size can be factored as $2^a \times 3^b \times 5^c \times 7^d$, where a, b, c, and d are non-negative integers, the library uses the Cooley-Tukey FFT algorithm [13]. For efficiency and consistency reasons, we will restrict our calculations to these sizes. It is also only necessary to look at a one-dimensional case, since it can be generalized to multiple dimensions as will be shown later.

The Cooley-Tukey FFT algorithm was developed by J.W. Cooley and John Tukey and released in their 1965 article "An algorithm for the machine calculation of complex Fourier series" [14]. The algorithm is based on factoring a N sized DFT to smaller DFTs of size N_1 and N_2.

First we denote

$$W = \exp(-2\pi i/N)$$

with which we can write the one-dimensional DFT compactly as

$$\mathcal{X}_n = \sum_{k=0}^{N-1} x_k W^{kn} \tag{2.36}$$

Calculating the equation above would require N^2 operations, which are defined by Cooley and Tukey to be a complex multiplication followed by a complex addition. Instead they suggest an algorithm with less than $2N\log_2 N$ operations as follows.

The number of points in the array that is being operated on should be a product of two factors, $N = N_1 N_2$. Now indices k and n can be written as

$$k = N_1 k_2 + k_1 \quad n_1, k_1 = 0, ..., N_1 - 1 \tag{2.37}$$

$$n = N_2 n_1 + n_2 \quad n_2, k_2 = 0, ..., N_2 - 1 \tag{2.38}$$

Then equation 2.36 can be written

$$\mathcal{X}_{N_2 n_1 + n_2} = \sum_{k_1=0}^{N_1-1} W^{N_2 k_1 n_1} W^{k_1 n_2} \sum_{k_2=0}^{N_2-1} x_{N_1 k_2 + k_1} W^{N_1 k_2 n_2} \tag{2.39}$$

$$\mathcal{X}_{N_2 n_1 + n_2} = \sum_{k_1=0}^{N_1-1} W_1^{k_1 n_1} W^{k_1 n_2} \sum_{k_2=0}^{N_2-1} x_{N_2 n_1 + n_2} W_2^{k_2 n_2} \tag{2.40}$$

where

$$W_{1,2} = \exp(-2\pi i / N_{1,2})$$

Looking at equation 2.40 it can be seen that a DFT of size N can be written as two DFTs with the introduction of *twiddle factors* $W^{k_1 n_2}$. We can then reduce the computation of $\mathcal{X}_n$ to three steps: (1) evaluating N_1 DFTs corresponding to values k_1, (2) multiplying this result by the twiddle factors and then (3) calculating N_2 DFTs of the DFTs in (1).

$$(1) \quad \mathcal{Y}_{k_1, n_2} = \sum_{k_2=0}^{N_2-1} x_{N_1 k_2 + k_1} W_2^{k_2 n_2} \tag{2.41}$$

$$(2) \quad \mathcal{Y}_{k_1, n_2} W^{m_1 k_2} = \sum_{k_2=0}^{N_2-1} x_{N_1 k_2 + k_1} W_2^{k_2 n_2} W^{m_1 k_2} \tag{2.42}$$

$$(3) \quad \mathcal{X}_{N_2 n_1 + n_2} = \sum_{k_1=0}^{N_1-1} \mathcal{Y}_{k_1, n_2} W^{m_1 k_2} W_1^{k_1 n_1} \tag{2.43}$$

It takes N_2 operations to calculate array $\mathcal{Y}_{k_1, n_2}$, which has N elements and which then results to $N_2 N$ operations. Similarly to calculate equation 2.43 we need $N_1 N$ operations, which adds to a total of $N(N_1 + N_2)$ operations for the whole process. This process can then be used successively m-times to give a total number of operations $N(N_1 + N_2 + ... + N_m)$, where $N = N_1 \cdot N_2 \cdot ... \cdot N_m$. It can be seen that if $N_d = s_d t_d$, then $s_d + t_d < N_d$ unless $s_d = t_d = 2$. This means that maximizing the number of steps gives the smallest number of operations, but factors of 2 are combined as pairs resulting in no loss. Setting all N_d equal to r will then result in

$$d = \log_r N$$

and a total number of operations

$$T(r) = rN\log_r N. \tag{2.44}$$

Cooley and Tukey continue to analyze in their article that $r = 3$ is formally the most efficient choice, but the use of $r = 2$ or $r = 4$ have advantages due to binary arithmetic used in addressing and in multiplication in computers. It has later been proven that their analysis was erroneous, and that larger $r = 32$ or $r = 64$ are better to efficiently utilize the registers in modern processors [15]. Which r is used is often described in the name of the FFT algorithm such as radix-2 or radix-3 with $r = 2$ and $r = 3$ respectively.

Cooley and Tukey also note, that since the multiplications inside the sums in equations 2.41 and 2.43 can be carried out simultaneously paralleling the FFT is permitted.

2.3.3 Multidimensional FFTs

In his book "Fast Fourier Transfrom and Convolution Algorithms" [12] H.J. Nussbaumer explains the multidimensional FFT as follows. For a two-dimensional array of size $N_1 \times N_2$ the DFT is

$$\mathcal{X}_{n_1,n_2} = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} x_{k_1,k_2} W_1^{k_1,n_1} W_2^{k_2,n_2} \tag{2.45}$$

using the same notation as in section 2.3.2. This equation can also be written as

$$\mathcal{X}_{n_1,n_2} = \sum_{k_1=0}^{N_1-1} W_1^{k_1,n_1} \sum_{k_2=0}^{N_2-1} x_{k_1,k_2} W_2^{k_2,n_2} \tag{2.46}$$

Reminiscent of the 1D DFT, we first evaluate N_1 DFTs of length N_2 for values k_1

$$\mathcal{Y}_{k_1,n_2} = \sum_{k_2=0}^{N_2-1} x_{k_1,k_2} W_2^{k_2,n_2} \tag{2.47}$$

Substituting this to equation 2.46 we see that we get another DFT

$$\mathcal{X}_{n_1,n_2} = \sum_{k_1=0}^{N_1-1} W_1^{k_1,n_1} \mathcal{Y}_{k_1,n_2} \tag{2.48}$$

This method is called the row-column method, since it can be seen as arranging the input into an array of size $N_1 \times N_2$ and setting the input data as the array's rows and columns. The DFTs are then computed first in columns and then in rows. A similar process can be seen also in the Cooley-Tukey FFT algorithm, as it splits an

one-dimensional array to two arrays, which could be thought of as a transform from one to two dimensions. Figure 2.3 visualizes an example of the row-column method.

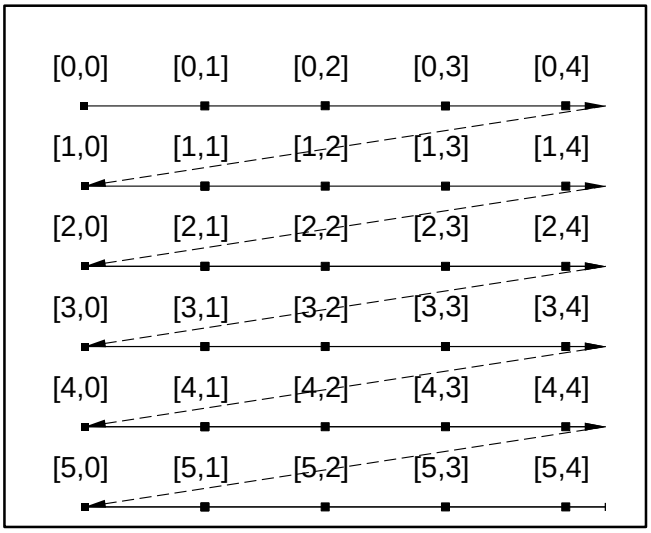


Figure 2.3: A 2D array represented as a 1D vector in a row-major order. Other possible implementation of the row-column method is the column-major order, where instead of $[0,0],[0,1],\ldots$ the order is $[0,0],[1,0],[2,0],\ldots,[0,1],[1,1],\ldots$

The process can be similarly repeated for any d-dimensional array.

2.4 FFT solution for Poisson's equation

Poisson's equation in three dimensions for electrostatic potential, V, and charge density, ρ, in a box with sides $\mathbf{L} = (L_1, L_2, L_3)$ is known to be

$$\nabla^2 V(\mathbf{x}) = \frac{-\rho(\mathbf{x})}{\epsilon_0} \qquad \mathbf{x} \in [\mathbf{0}, \mathbf{L}] \tag{2.49}$$

where $\mathbf{x} = (x_1, x_2, x_3)$, ϵ_0 is the vacuum permittivity and ∇^2 is the Laplace operator, which we now write as

$$\nabla^2 = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \frac{\partial^2}{\partial x_3^2}. \tag{2.50}$$

We will consider the periodic boundary conditions where

$$V(0,x_2,x_3)=V(L_1,x_2,x_3)=0 \qquad \frac{\partial}{\partial x_1}V(0,x_2,x_3)=\frac{\partial}{\partial x_1}V(L_1,x_2,x_3)=0$$
$$V(x_1,0,x_3)=V(x_1,L_2,x_3)=0 \qquad \frac{\partial}{\partial x_2}V(x_1,0,x_3)=\frac{\partial}{\partial x_2}V(x_1,L_2,x_3)=0$$
$$V(x_1,x_2,0)=V(x_1,x_2,L_3)=0 \qquad \frac{\partial}{\partial x_3}V(x_1,x_2,0)=\frac{\partial}{\partial x_3}V(x_1,x_2,L_3)=0$$

since we know that the potential decreases with distance and $\mathbf{L}$ can be set large enough to fulfill these conditions.

First we take a sample of $\mathbf{N}=(N_1,N_2,N_3)$ points of the Fourier series of V and ρ and substitute them to equation 2.49. While sampling we will have to choose points of $\mathbf{x}$ within the sample, which means setting $\mathbf{x}=(n_1\Delta x_1,n_2\Delta x_2,n_3\Delta x_3)$, with $\Delta x_d=\frac{L_d}{N_d}$ and $n_d\in\mathbb{N}$ for $d=1,2,3$. This sampling is done to emphasize the connection of this solution with the FFT, but the solution could be alternatively achieved with the use of Fourier transforms property for its derivatives $\mathcal{F}\left(\frac{\mathrm{d}^n}{\mathrm{d}x^n}f(x)\right)=(-2\pi ik)^n\widetilde{f}(k)$. We will continue with the same notation as in section 2.3.1, where $\frac{\mathbf{x}}{\mathbf{X}}=(\frac{x_1}{X_1},\frac{x_2}{X_2},\frac{x_3}{X_3})$. Thus we get

$$\nabla^2\left(\sum_{\mathbf{k}=-\infty}^{\infty}\widetilde{V}(\mathbf{k})\exp\left(-2\pi i\frac{\mathbf{kx}}{\mathbf{L}}\right)\right)=\frac{-1}{\epsilon_0}\sum_{\mathbf{k}=-\infty}^{\infty}\widetilde{\rho}(\mathbf{k})\exp\left(-2\pi i\frac{\mathbf{kx}}{\mathbf{L}}\right) \tag{2.51}$$
$$\Rightarrow\nabla^2\left(\sum_{\mathbf{k}=0}^{\mathbf{N}-1}\widetilde{V}(\mathbf{k})\exp\left(-2\pi i\frac{\mathbf{kn}}{\mathbf{N}}\right)\right)=\frac{-1}{\epsilon_0}\sum_{\mathbf{k}=0}^{\mathbf{N}-1}\widetilde{\rho}(\mathbf{k})\exp\left(-2\pi i\frac{\mathbf{kn}}{\mathbf{N}}\right) \tag{2.52}$$

and with chain rule for derivatives the partial derivatives in the Laplacian operator change to

$$\frac{\partial^2}{\partial x_d^2}=\frac{\partial}{\partial x_d}\frac{\partial}{\partial x_d}=\frac{\partial}{\partial x_d}\left(\frac{\partial}{\partial n_1}\frac{\partial n_1}{\partial x_d}\right)=\frac{1}{\Delta x_d}\frac{\partial}{\partial x_d}\frac{\partial}{\partial n_d} \tag{2.53}$$
$$=\frac{1}{\Delta x}\frac{\partial}{\partial n_d}\left(\frac{\partial n_d}{\partial x_d}\frac{\partial}{\partial n_d}\right) \tag{2.54}$$
$$=\frac{1}{\Delta x_d^2}\frac{\partial^2}{\partial n_d^2} \tag{2.55}$$

Due to Laplacian's linearity the derivatives on the LHS of equation 2.52 can be taken separately for each of the terms resulting in

$$-\sum_{\mathbf{k}=0}^{\mathbf{N}-1}(2\pi)^2\left(\left(\frac{k_1}{\Delta x_1N_1}\right)^2+\left(\frac{k_2}{\Delta x_2N_2}\right)^2+\left(\frac{k_3}{\Delta x_3N_3}\right)^2\right)\widetilde{V}(\mathbf{k})\exp\left(2\pi i\frac{\mathbf{kn}}{\mathbf{N}}\right)$$
$$=\frac{-1}{\epsilon_0}\sum_{\mathbf{k}=0}^{\mathbf{N}-1}\widetilde{\rho}(\mathbf{k})\exp\left(2\pi i\frac{\mathbf{kn}}{\mathbf{N}}\right) \tag{2.56}$$

$$-\sum_{\mathbf{k}=0}^{\mathbf{N}-1}(2\pi)^2\left(\left(\frac{k_1}{L_1}\right)^2+\left(\frac{k_2}{L_2}\right)^2+\left(\frac{k_3}{L_3}\right)^2\right)\widetilde{V}(\mathbf{k})\exp\left(2\pi i\frac{\mathbf{kn}}{\mathbf{N}}\right)$$
$$=\frac{-1}{\epsilon_0}\sum_{\mathbf{k}=0}^{\mathbf{N}-1}\widetilde{\rho}(\mathbf{k})\exp\left(2\pi i\frac{\mathbf{kn}}{\mathbf{N}}\right) \quad (2.57)$$

The coefficients of the exponent term are all independent of each other, which gives an equation

$$-(2\pi)^2\left(\left(\frac{k_1}{L_1}\right)^2+\left(\frac{k_2}{L_2}\right)^2+\left(\frac{k_3}{L_3}\right)^2\right)\widetilde{V}(\mathbf{k})=\frac{-\widetilde{\rho}(\mathbf{k})}{\epsilon_0} \quad (2.58)$$
$$\Rightarrow \widetilde{V}(\mathbf{k})=\frac{\widetilde{\rho}(\mathbf{k})}{\epsilon_0(2\pi)^2\left|\frac{\mathbf{k}}{\mathbf{L}}\right|^2} \quad (2.59)$$

where the terms in the denominator have been collected together as their Euclidean length squared. By taking the iDFT we will get a part of the solution for potential

$$\mathcal{F}^{-1}\left[\widetilde{V}(\mathbf{k})\right]=\mathcal{F}^{-1}\left[\frac{\widetilde{\rho}(\mathbf{k})}{\epsilon_0(2\pi)^2\left|\frac{\mathbf{k}}{\mathbf{L}}\right|^2}\right](\mathbf{n}) \quad (2.60)$$
$$V(\mathbf{n})=\mathcal{F}^{-1}\left[\frac{\widetilde{\rho}(\mathbf{k})}{\epsilon_0(2\pi)^2\left|\frac{\mathbf{k}}{\mathbf{L}}\right|^2}\right](\mathbf{n}) \quad (2.61)$$

This solution is defined to only within a constant C, which can be seen by substituting the solution to the intermediate result 2.52. We also notice a singularity when all the components of $\mathbf{k}$ are zero, but this term is actually included in the constant C, as we can look at $\mathbf{k}=\mathbf{0}$ individually from equation 2.52 and see that it is a constant term removed by the Laplacian operator.

Looking at our boundary conditions and assuming that the system is charge neutral, we notice that C can be set to zero and so the contribution of $\mathbf{k}=\mathbf{0}$ is also zero, and the equation for potential can be kept in the form of equation 2.61. For a non-neutral system the RHS of 2.52 will be non-zero, which can be seen by substituting the inverse Fourier transformation of $\widetilde{\rho}$ to it. This is why the assumption of neutral charge is required.

It should also be noted, that if we had chosen to use the aforementioned Fourier transforms property for its derivatives, we would have to check our DFT's convention with regards to $\mathbf{k}$, as it can vary. This solution is specific to the convention described in section 2.3.1.

3. Methods

3.1 Parallel computing

A common method of performing computations on personal computers is doing the computations in *serial*. In this kind of computation the operations could be thought as a queue which get calculated one at a time after each other. For example simple sum $1+2+3+4$ could be seen in serial as $1+2+3+4 = 3+3+4 = 6+4 = 10$, which would take three operations and a time of $3 \cdot$ (time to sum two numbers). An alternative to serial computing is *parallel* or *distributed* computing, where the larger computation is spread to smaller tasks which are then independently calculated by different processors. Continuing the sum example, we could split the task so that processor 1 does $1 + 2$ while at the same time processor 2 does $3 + 4$ then receives the result of processor 1 and gets the final result $7+3 = 10$. Although the number of operations stays the same, the time spent is now 2/3 of the serial counterparts time, as the first two steps of the parallel computation were done simultaneously.

This example is highly idealized and simplified, as from it one could think that everything is better done in parallel. This isn't the case as our computations are limited by the architectures of our computers, and so we have to consider three relevant issues when considering parallelizing our computations. In their book "Parallel and Distributed Computation Numerical Methods" [16] Dmitri Bertsekas and John Tsitsiklis summarize these issues to be *task allocation, communication* and *synchronization*. To keep this topic concise we will not discuss computer architecture at a detailed level, rather we will try to give a basic understanding of the uses and limitations of parallel computing.

Task allocation describes itself quite well. If the task at hand cannot be allocated efficiently to a parallel architecture, the serial counterpart will likely outperform it. Consider a similar example to our previous one, where we have to add together several large numbers. These large numbers could be too large to store in the local memory of our parallel processors, meaning the task would have to be spread to smaller subtasks. This task allocation might in turn take more time or prove to be more complex than doing the calculations in serial. Also as computations and algorithms get more

complicated there might not be a clear way to formulate task allocation.

Going once again back to our first example, if we want to do the calculation our processors have to communicate between each other during the last step. Here we have the local memory of both processors, and we have to move a variable from one to the other. If moving this variable takes more time than doing the calculation of it independently on the receiving processor, it would naturally be faster just to do it there instead of waiting for the variable to arrive. The final address of the calculations isn't necessarily the address where the variable is stored or used next, which should also be taken into account. We also mentioned distributed systems, which Bertsekas and Tsitsiklis [16] describe as "loosely coupled; there is very little, if any, central coordination and control," compared to parallel systems which "consist of several processors that are located within a small distance of each other." It is understandable, that the role of efficient communication is significant when considering distributed systems. Communication in parallel systems can be improved by using shared memory, which means the processors have a memory space they share, but this might lead to slower calculations than with strictly local memories.

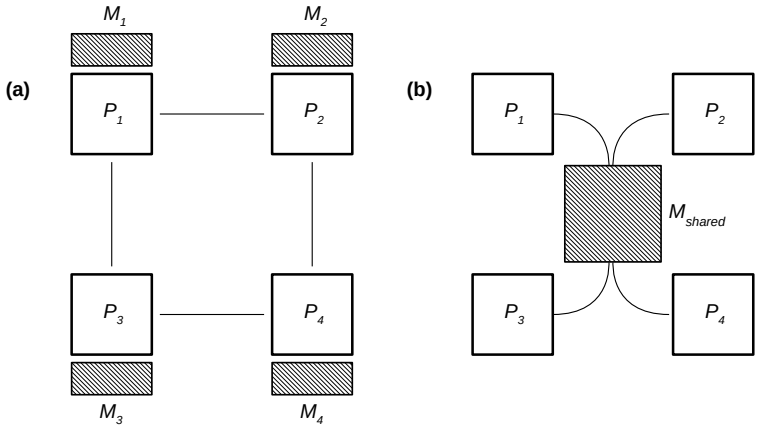


Figure 3.1: Local memory **(a)** and shared memory **(b)** implementations. Adapted from [16] Figure 1.1.3.

Lastly there's the issue of synchronization. In a parallel algorithm it is required that the activities of different processors are coordinated to some extent. For implementing this coordination an algorithm can be split to phases, where a processor does computations independent of other processor, but all the processors are in the same phase of the algorithm. Communication between processors happens between these

phases. This kind of algorithm is called synchronous and an algorithm without phase-implementation is called asynchronous. Synchronous algorithms can be implemented as globally synchronized, where a global clock communicates with the computing processors the current phase, or algorithms can be implemented locally synchronized, where each processor knows beforehand after which result it should start a new phase. Due to the communication aspect of synchronous algorithms they are limited by communication penalty, communication between phases takes time which is then taken from the calculations. A faster processor is also limited by slower processors in synchronous algorithms, as it has to wait for the slower processor to reach the end of each phase before continuing. In asynchronous algorithms processor don't wait for intermediate results or phases of other processors, but this comes at the cost that some algorithms require these intermediate results and thus aren't implementable as asynchronous. Asynchronous algorithms are often used in data networks, where each node of the network has to operate independently as the condition of links between nodes or other nodes shouldn't be allowed to affect the individual performance of a node.

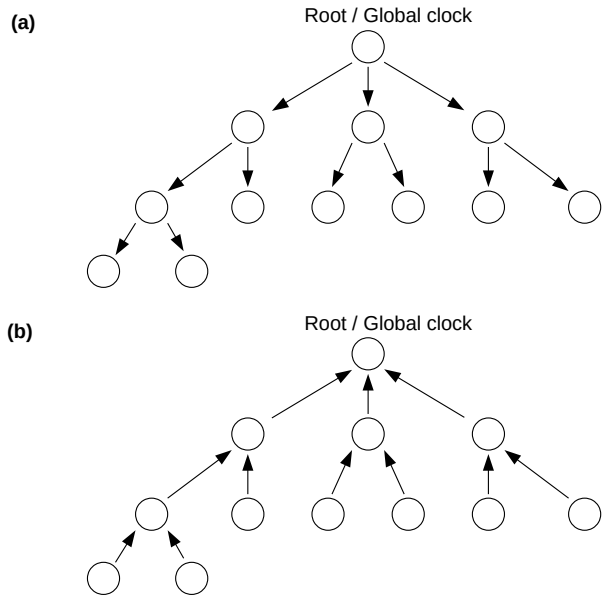


Figure 3.2: A synchronous algorithm where a root node propagates **(a)** and terminates **(b)** a phase. Adapted from [16] Figure 1.4.2.

3.1.1 Implementing parallel computing

Parallel computing can be implemented through the use of either programming languages designed for parallel computing or platforms and models that use existing programming languages to implement parallelization. Our code uses NVIDIA's CUDA

model for C, C++ and Fortran for parallelization, with electron density calculations implemented in C++ and the electrostatic potential calculations in Fortran. In CUDA the developer creates code for a host (CPU) and a device (GPU) [17]. Host code is used to copy variables between the host and the device and to coordinate the process, while device code is more focused on doing the calculations.

To better understand the device code and differences between CPU and GPU code, understanding their architecture is essential. CPUs are designed to execute sequences of operations (threads) as fast as possible, with possibilities of executing few tens of threads in parallel. In contrast, GPUs are designed to execute thousands of threads in parallel. This means that GPUs architecture and resources are more focused on processing data instead of caching and controlling the flow of it [17].

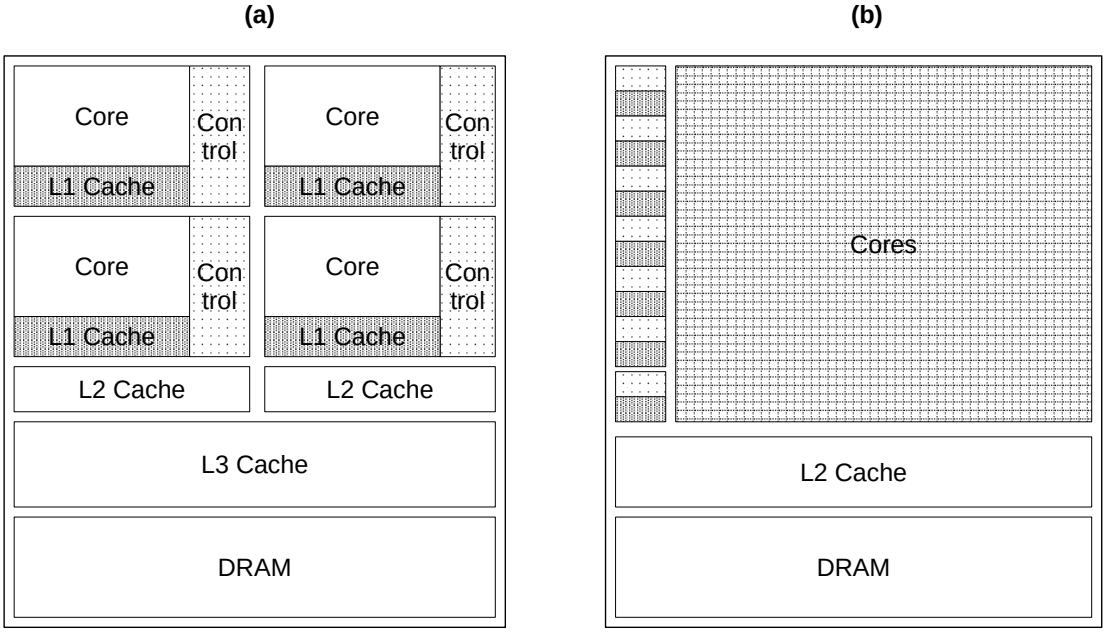


Figure 3.3: Exmple architectures for a **(a)** CPU and **(b)** GPU. Adapted from [17] Figure 1.

Instructions for the device are done in "kernels." Kernels define what a single thread does and they are called for all the threads defined in execution configuration. A single thread is processed by a thread processor (core), multiple threads form a thread block which is processed by a multiprocessor. These thread blocks from a grid, which represents the hardware device in the software. Execution configuration defines the number blocks in a grid and the number of threads in a block. When programming, hardware limitations should be taken into account when choosing the number of threads

and blocks.

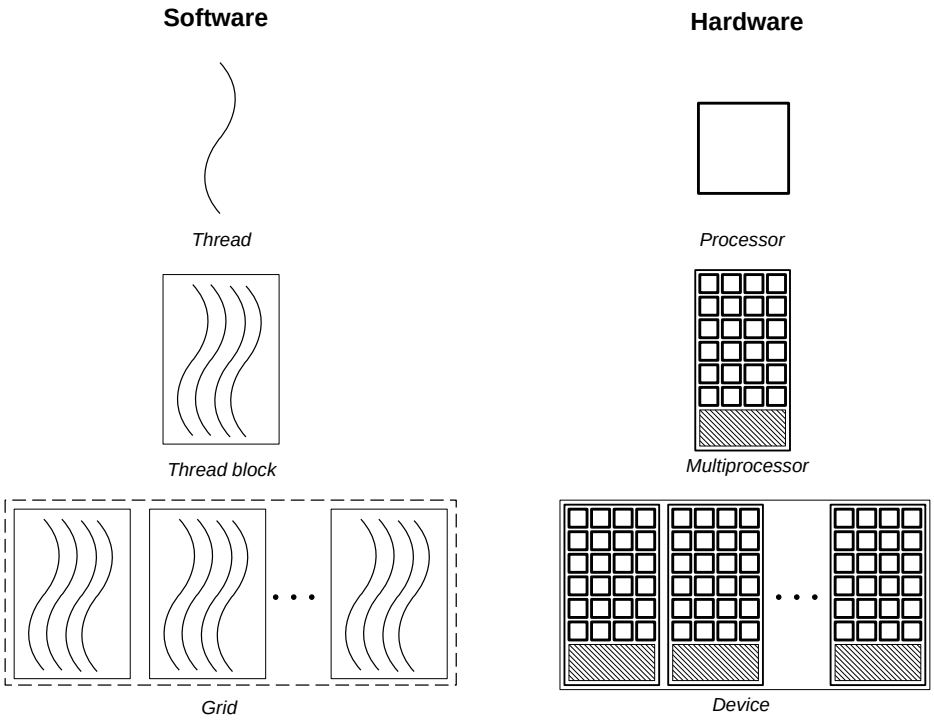


Figure 3.4: Software properties and their hardware counterparts. Adapted from [17] Figures 5. and 6.

Problems that can easily be decomposed along spatial dimension are ideal for parallel computing [16]. In our case we have a grid where each point represents a value at that location. By assigning a thread for each of these grid points we can do calculations for them simultaneously instead of going through all the points in serial. For larger systems a thread might need to be given multiple points due to hardware limitations, but this limit hasn't been tested for our project.

3.2 Calculating the electrostatic potential

The following process was used to calculate the electrostatic potential for a molecule with known nuclear positions:

1. Initialize grid for the given molecule (see Figure 3.5).
   (a) Choose a grid precision (step). This specifies the distance between points in the grid.
   (b) Choose the amount of empty space around the molecule (fat). This specifies the distance between the outermost nucleus' and the sides closest to them.

2. Use the coupled clusters method to retrieve the electron densities in the grid – electron densities calculated through other methods can also be used.
3. Calculate the total charge at each point of the grid.
   (a) Change from electron density to electron charge density.
   (b) Add the contribution of nuclear charge density to the electron charge density.
4. Perform DFT on the grid.
5. Calculate the potential in the frequency domain.
6. Perform inverse DFT on the grid to retrieve the result.

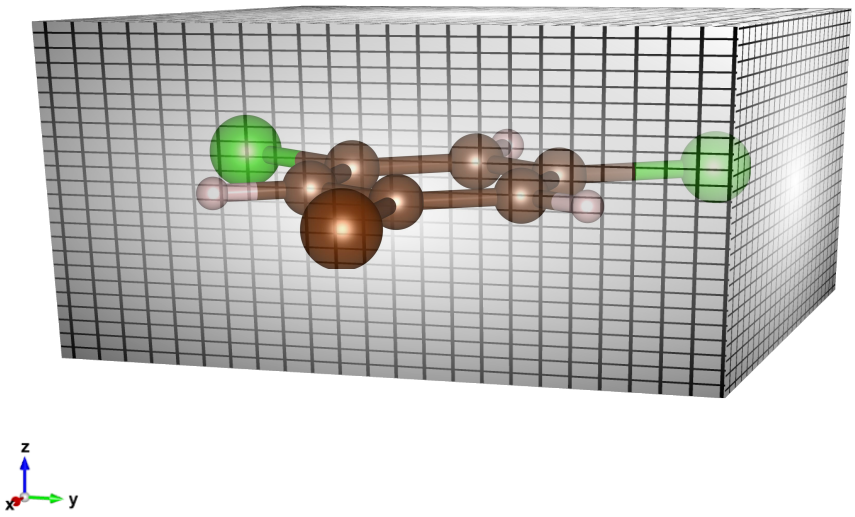


Figure 3.5: Bromodichlorobenzene in a grid.

3.2.1 Spread of nuclear charge

At step 3b of our process we have to add the charge density of the nuclei to our grid. Due to the much higher charge that the nuclei have compared to electrons, this addition will result in a spike in the values of the array. Although physically feasible and a finite value, this value will negatively affect the accuracy of the DFT, since

discontinous or functions that aren't very smooth require more Fourier coefficients to represent them and the result might exhibit Gibbs phenomenon, an overshoot of values near the locations of the discontinuities [18]. We offer two solutions to increase the smoothness near the nuclei, one that convolves the charge to neighbouring points and one that assumes a static spherically symmetric Gaussian charge density near the nuclei.

Convolving the charge

Initially we find the location of a nucleus in a $2 \times 2 \times 2$ grid inside the total grid and spread it linearly to the points in the smaller grid according to it's distance to each point. For this we calculate the array indices of the nucleus in the total grid, and make this our origin (0,0,0) for the smaller grid. Then we calculate and compare the position of the nucleus to the position of our smaller grids origin in the total grid, resulting in a variable $\mathbf{d} = (d_1, d_2, d_3)$ which has values from 0 to 1. Small values for $|\mathbf{d}|$ mean the nucleus is close to the origin of the smaller grid. Each of the points in the smaller grid is then given a percentage of the charge according to $\mathbf{d}$. This is repeated for all the atoms in the molecule.

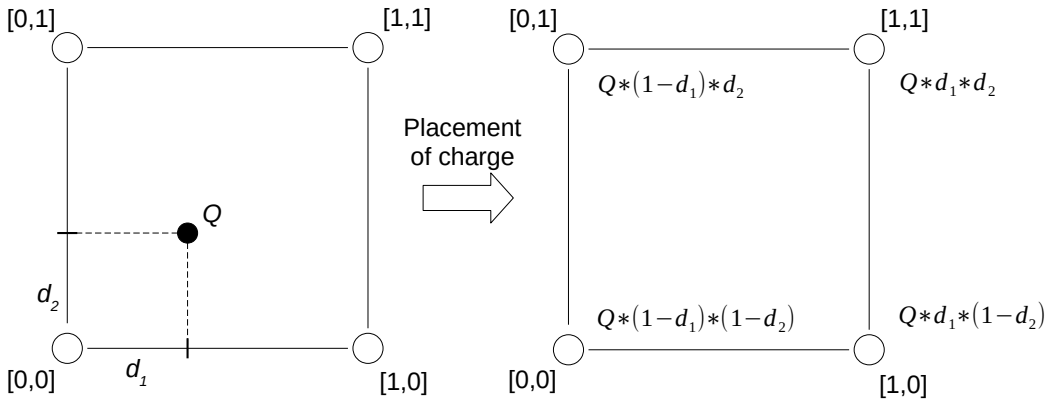


Figure 3.6: Placement of charge Q to its neighbouring points in two dimensions.

After the charge has been placed we start convolving it with some convolution

constant D and some number of convolutions N_D. For each point in our total grid a point gains some charge according to D, the charge in its neighbouring points and the points distance to its neighbouring points. To compensate the charge gained from neighbouring points, the point also loses charge according to the same variables. This is done so that the total charge on the grid is conserved.

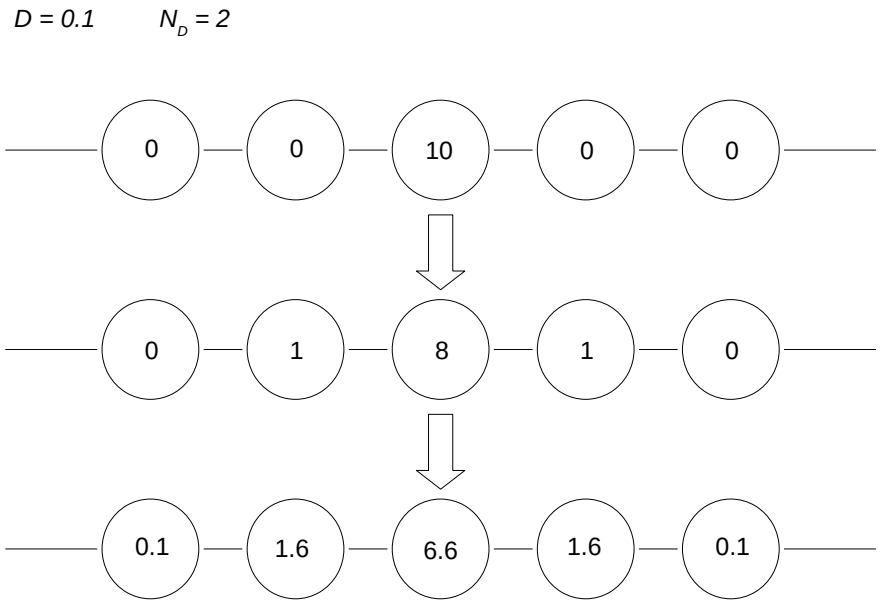


Figure 3.7: One dimensional example of convolving values in a grid.

Gaussian charges

By assuming static spherically symmetric Gaussian charge density, the charge density at point $\mathbf{r}$ due to nucleus at point $\mathbf{r_0}$ with charge Q can be calculated as

$$\rho(\mathbf{r}) = \frac{Q}{\sigma^3 \cdot (2\pi)^{3/2}} \exp\left(-\frac{|\mathbf{r} - \mathbf{r_0}|^2}{(2\sigma^2)}\right) \tag{3.1}$$

If calculated for all points in the grid, this would result in an unnecessary amount of calculations and memory usage. Instead it can be shown that the effect of a Gaussian at distance of more than 3σ is approximately zero (see Figure 3.8), which allows us to limit the points to a $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$ grid around the positions of the nuclei. Our variance is the step size chosen in 1a of the total process.

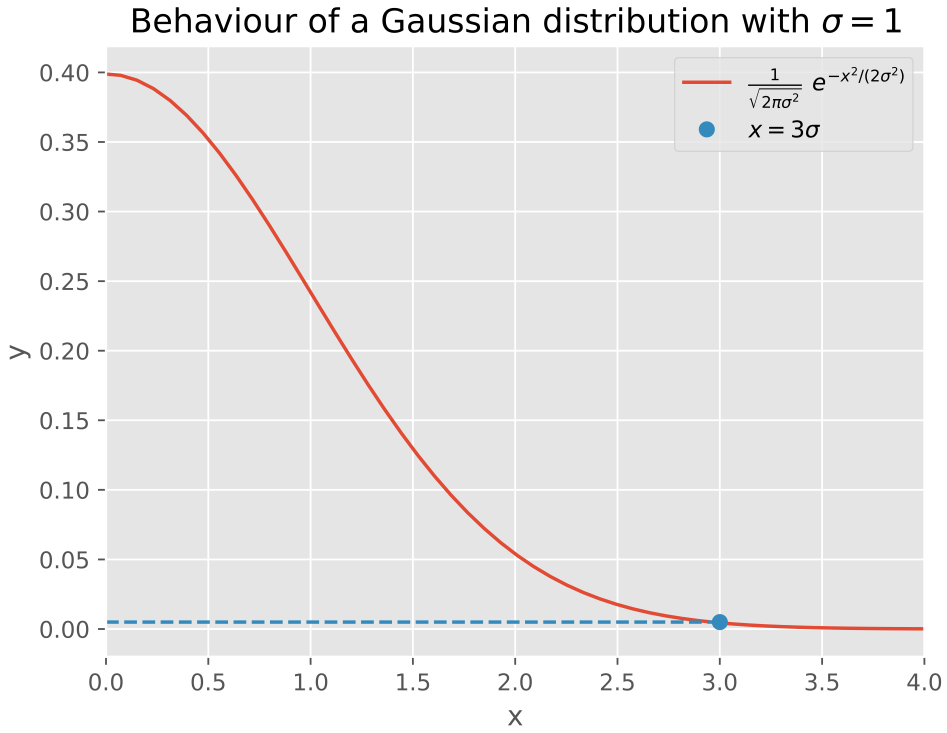


Figure 3.8: Behaviour of a Gaussian distribution.

4. Results

4.1 Computing a molecule's electrostatic potential

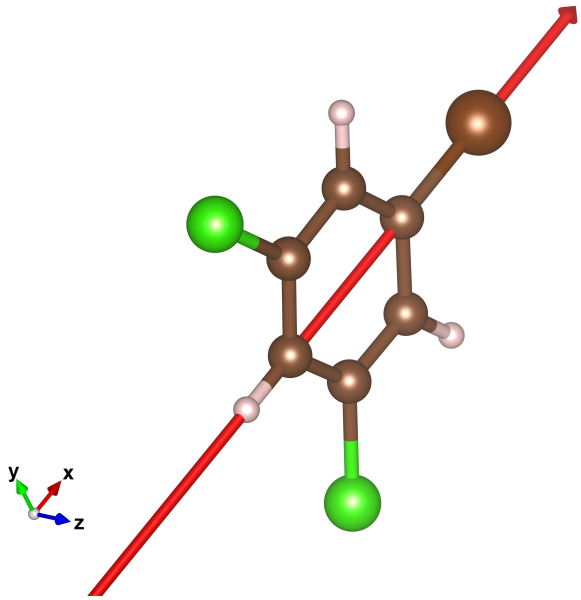


Figure 4.1: Orientation of a bromodichlorobenzene molecule and a path through it used for the analysis of the calculations. Atoms on the path from left to right: hydrogen (light pink), carbon (light brown), carbon and bromine (brown). Green atoms outside the path are chlorine.

We chose to look at a bromodichlorobenzene (BCB) molecule for testing the calculations, as it is a relatively simple molecule but contains various elements. Figure 4.2 shows the output electrostatic potential produced for the test molecule at varying heights. From it the electrostatic potential can be seen behaving as it should, decreasing with distance from the atoms, and that the overall mapping doesn't contain any spikes or values that are out of place.

To have a better look at the accuracy of our method, we will look at it's behaviour while approaching and passing the nuclei of our example molecule, BCB. Figure 4.3 shows values produced by our code while passing through the nuclei of the molecule as shown in Figure 4.1. The positions of the nuclei can be seen in the values of the

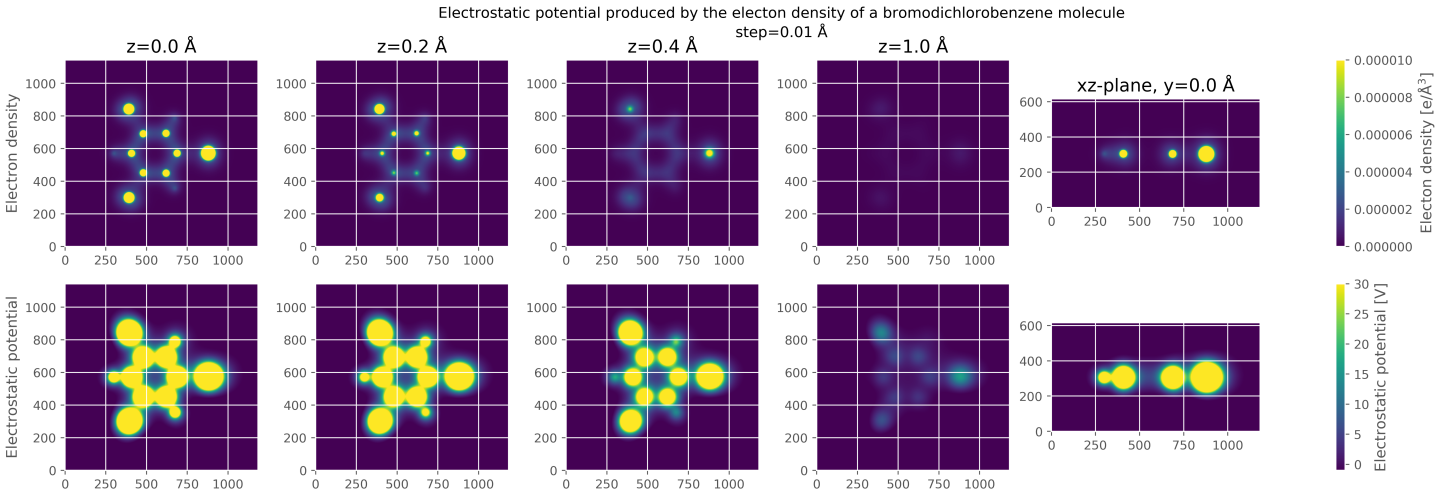


Figure 4.2: Electrostatic potential of a bromodichlorobenzene and the electron density from which it was calculated. Units for all axes are Å/step. Origin is set to the middle of the grid. Vertical axes correspond to the x-axis of Fig. 4.1 and the horizontal axis to the y-axes and the z-axis for the last image in this figure. Brightest areas contain values equal to or higher than the maximum value of the colorbar, darkest ares have the same behaviour with values equal to or less than the minimum value.

potential as spikes, since it is known that the value for the electrostatic potential approaches infinity while approaching the position of a nucleus. Bonding between the atoms can also be recognized from the electrostatic potential, as the higher electron density causes a decrease in the electrostatic potential, but this can also be attributed to the distance between the nuclei. These properties show that this method for calculating the electrostatic potential gives physically reasonable results in the big picture, but behaviour of the smaller values or the values between the nuclei is still questionable.

Figure 4.3 shows that decreasing the step size will result in higher values for the whole system. At the positions of the nuclei this is reasonable, since a decreased step size should result in a higher accuracy and thus a better approximation of the $V \to \infty$ behaviour described above. Decreasing the step size also retains the behaviour that at the positions of bonds there is a clear decrease in the value for the electrostatic potential, but it can be seen from Figure 4.3b that the values at these positions are notably higher for step size 0.01 Å when compared to other step sizes. High values for the electrostatic potential inside the molecule might be correct, or these high values between the nuclei might be due to the Fourier transform not being accurate enough which could then result in the aliasing of values from the singularity to the surrounding space. Aliasing happens when frequencies outside of the range of sampled frequencies are included in the sampled frequencies [18]. Figure 4.3b also shows behaviour which could indicate erroneous methods or implementation, as it doesn't follow the $V(\mathbf{0}) = V(\mathbf{L}) = 0$ boundary conditions, although it still has periodicity ($V(\mathbf{0}) = V(\mathbf{L})$).

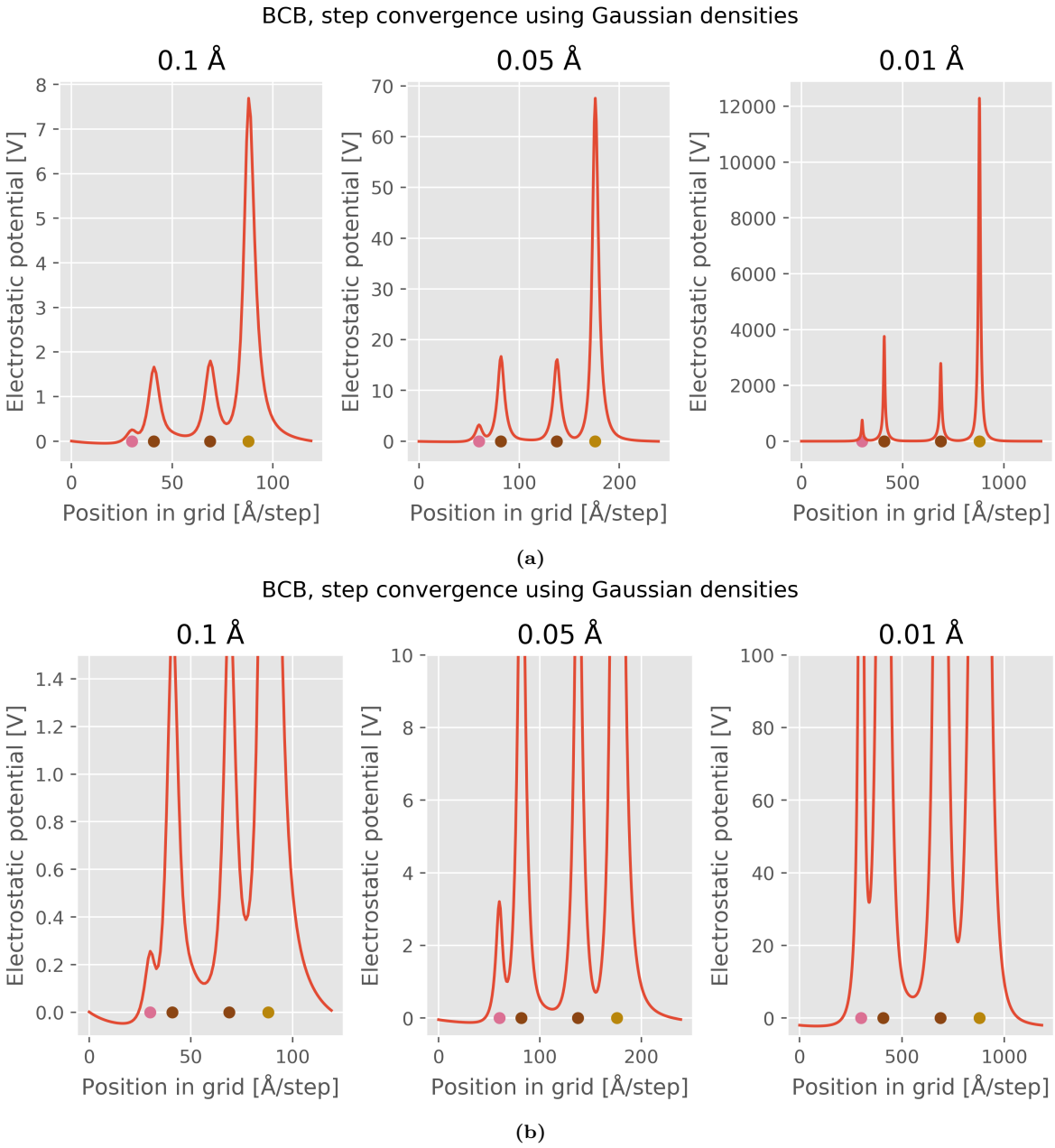


Figure 4.3: Values for electrostatic potential when passing through the hydrogen, carbon and bromine atoms in a bromodichlorobenzene molecule, as shown in Figure 4.1, with subfigure **(b)** having zoomed in y-axes. Gaussian densities were used for the nuclear charges.

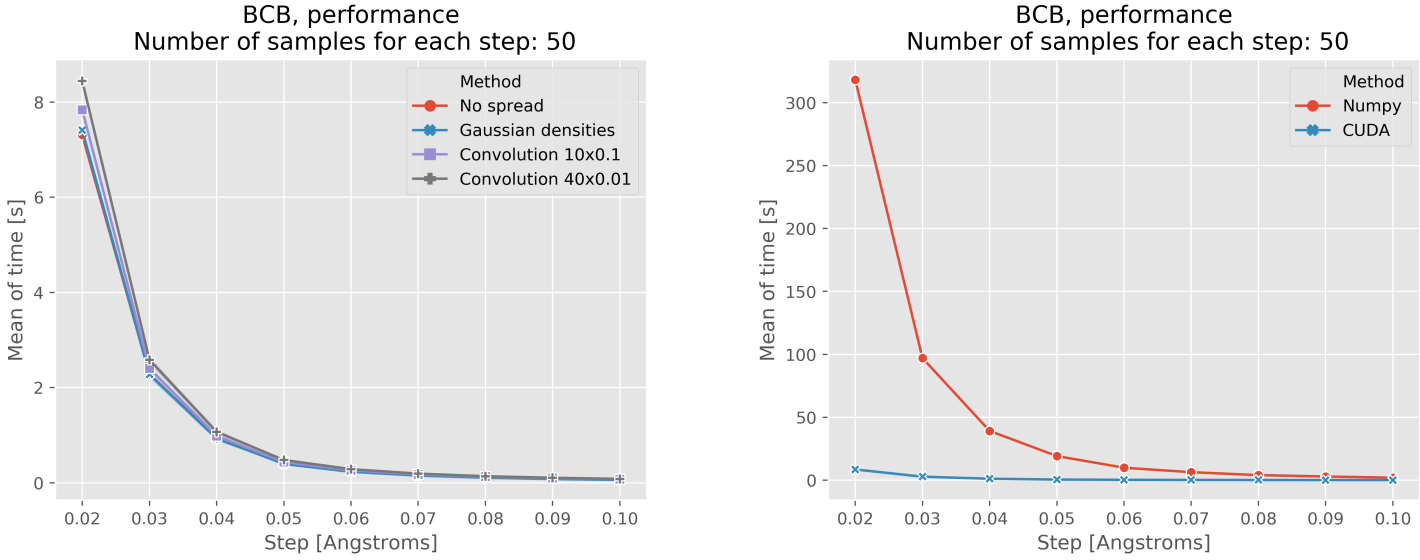
Other methods for the same situation showed similar results, as is shown in Figures A.2a and A.1a in Appendix A. Interestingly, Gibb's phenomenon didn't appear in the calculations with no spreading, and the results remained smooth everywhere outside the nuclei regardless of the chosen step size. Gibb's phenomenon can be forced or mistakenly caused by the use of bad convolution constants, as is shown in Figure A.5a. Convolution method shows lower values at the positions of the nuclei when

compared to other methods, but this can be explained by the nuclei losing charge to the surrounding points – loss of charge isn't as large in the Gaussian densities method. No other differences were found between the spreading methods as is shown in Figure A.3. Behaviour of the different spreading methods was identical between the nuclei, which is expected as the spreading methods only affect the space near the nuclei.

A similar test was done for a simpler dihydrogen molecule with plots of the results included in Appendix A. Similar behaviour was found with the dihydrogen as with the BCB.

4.2 Performance of different methods and implementations

Figure 4.4a shows the difference in performance between the methods used for spreading the charge. Leaving the charge unspread and using the Gaussian densities produce similar times, with the convolution methods using slightly more time. This is expected, as the convolution method has to perform multiple calculations for each point in space and to move results between the calculations – convolutions can't be performed asynchronously as they require information from the neighbouring points. This performance loss is something to consider if more convolutions are wanted or a step size smaller than 0.02 Ångströms is used.



(a) **(b)**

Figure 4.4: Performance comparisons of the spreading methods **(a)** and of the serial (Numpy) and parallel (CUDA) implementations **(b)**

A serial implementation of the code was done in Python with Numpy's FFT library and with the use of the Gaussian densities method for spreading the charge, although in this implementation the Gaussian distribution isn't limited as is described in section 3.2.1. This performance loss can be considered compensated through the use of Numpy's highly optimized code for performing vectorized calculations. Numpy's performance was then compared to its CUDA counterpart with results shown in Table 4.1 and visualized in Figure 4.4b. CUDA is seen to perform around 35 times faster than Numpy.

Step [Å] (Number of points)	Method	Mean of time [s]	Error of mean [± s]
0.1 (921600)	Numpy	1.926	0.0090
	CUDA	0.074	0.0063
0.09 (1253376)	Numpy	2.89	0.018
	CUDA	0.090	0.0010
0.08 (1751040)	Numpy	4.01	0.031
	CUDA	0.121	0.0012
0.07 (2601984)	Numpy	6.33	0.039
	CUDA	0.105	0.0019
0.06 (3993600)	Numpy	9.9	0.23
	CUDA	0.265	0.0020
0.05 (7127040)	Numpy	19.2	0.23
	CUDA	0.477	0.0041
0.04 (13639680)	Numpy	39.1	0.23
	CUDA	1.12	0.023
0.03 (31948800)	Numpy	97.0	0.37
	CUDA	2.75	0.021
0.02 (106389504)	Numpy	318.3	0.46
	CUDA	8.48	0.020

Table 4.1: Time to calculate BCB's electrostatic potential in serial (Numpy) and in parallel (CUDA) with Gaussian densities, each step has a sample size of 50. Number of points in the grid is shown in parenthesis after the step.

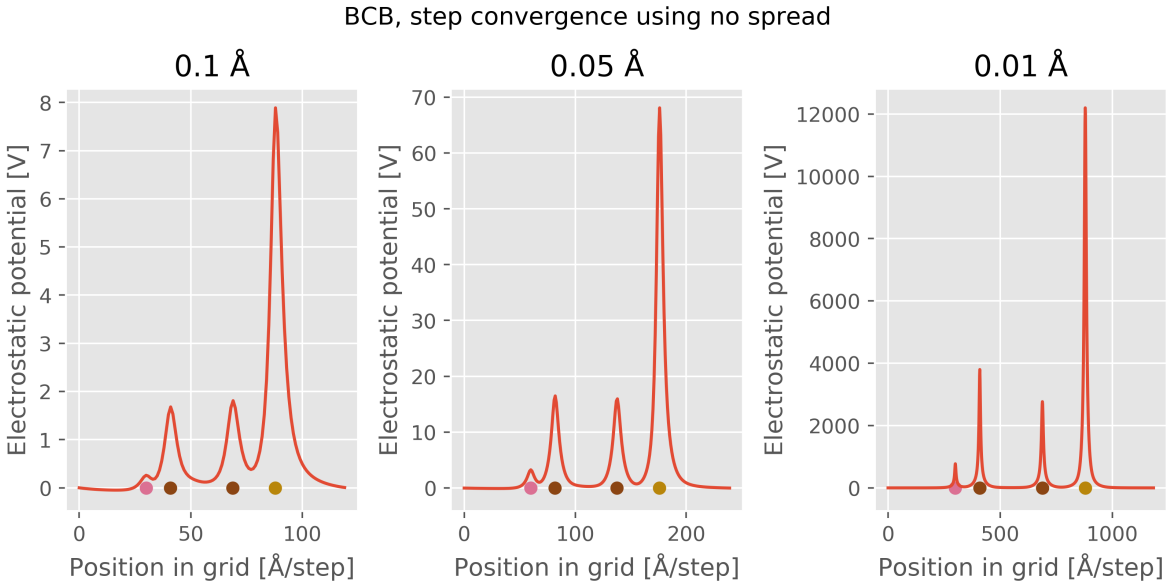
5. Conclusions

Results produced by my calculations were reasonable, as they showed some fundamental properties of the system. Inconsistencies could be seen in the boundary values and the values inside the system at hand, with possible aliasing of values near the nuclei recognized. For further analysis of the aliasing of values, the range of frequencies and the sample frequency of the system could be looked at, since through those variables aliasing can be identified [18]. Overall, the inconsistencies are difficult to analyse without a clear point of reference such as experimental data or a different method for calculating the electrostatic potential. There is also a fundamental problem when calculating the electrostatic potential for a non-periodic system using a periodic method, and the FFT solution might be more suitable for different molecules than what was tested for. A similar method as the FFT could possibly be implemented with the use of other basis functions such as Chebyshev or Legendre polynomials to match the non-periodic boundary conditions [18]. The FFT calculations were done using a single floating point precision, and more precise calculations could be done by using double precision, but this would likely be a minor improvement in precision leaving the accuracy unchanged.

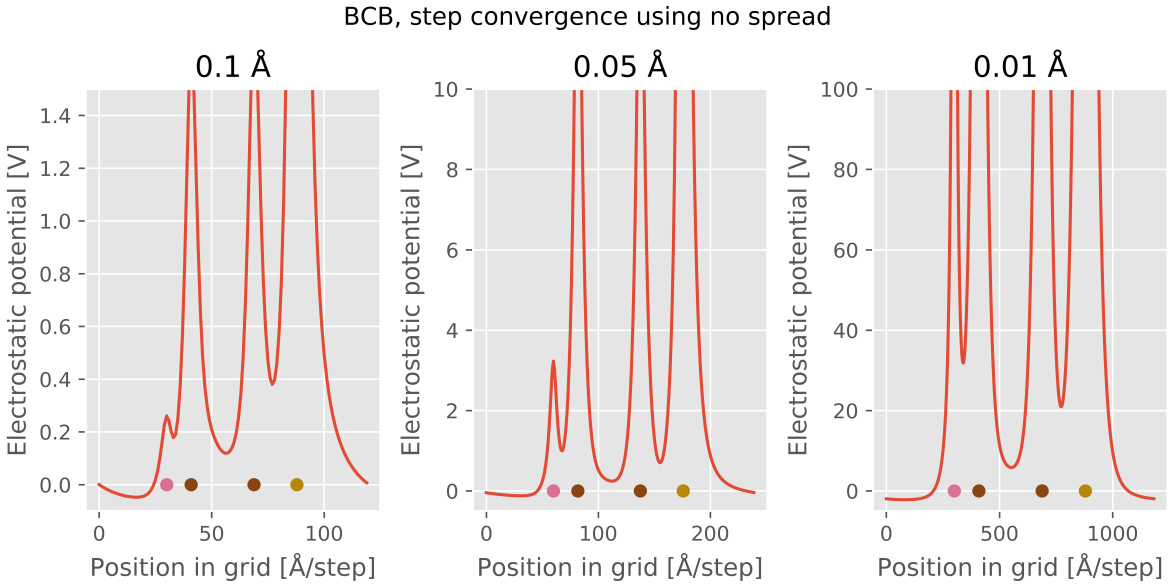
Spreading of the nuclear charge showed no particular improvement, as the non-spread calculations provided similar results as the spread calculations with faster performance. Out of all the presented methods either no spread or Gaussian densities should be used, as the convolution method showed no clear upsides compared to the other two methods but includes the risk of accidental Gibb's phenomenon. Convolving the charges could prove to be useful if the convolution constants could be chosen so that the output matches a reference result, after which the same constants could be used to analyse a different system. Accurate convolution constants could be found by looking for a relation between the constants and the number of points or the charges in the system.

It was also seen that the parallel implementation of the solution outperformed the serial implementation, as could be expected. The parallel version performed around 35 times faster than the serial version, but this could be improved further with better knowledge of CUDA and computer architecture. Nonetheless it is a good showcase of improving performance with relatively simple parallelization.

Appendix A. Figures

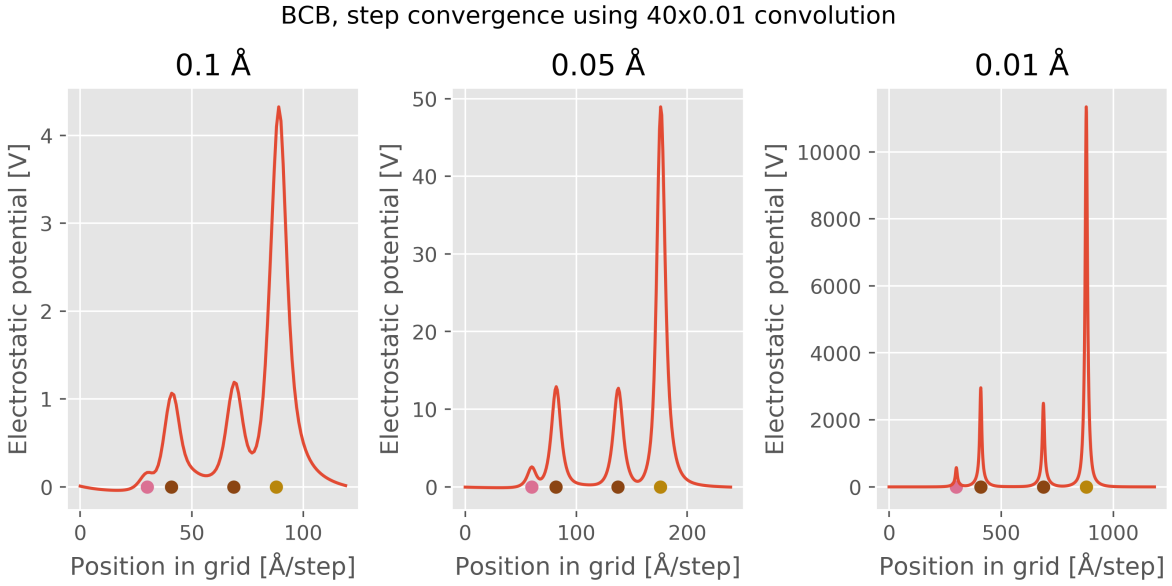


(a)

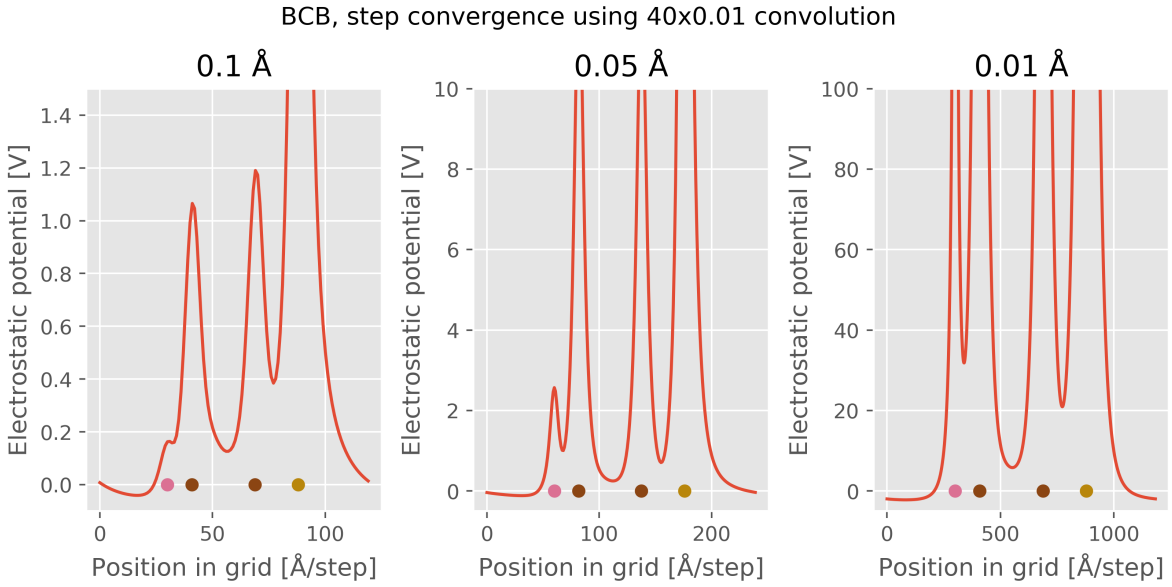


(b)

Figure A.1: Values for electrostatic potential when passing through the hydrogen, carbon and bromine atoms in a BCB molecule for different step sizes, as shown in Figure 4.1, with subfigure **(b)** having zoomed in y-axes. No spreading was used for the nuclear charges.



(a)



(b)

Figure A.2: Values for electrostatic potential when passing through the hydrogen, carbon and bromine atoms in a BCB molecule, as shown in Figure 4.1, with subfigure **(b)** having zoomed in y-axes. Convolving with constants N_d=40 and $D = 0.01$ was used for the nuclear charges.

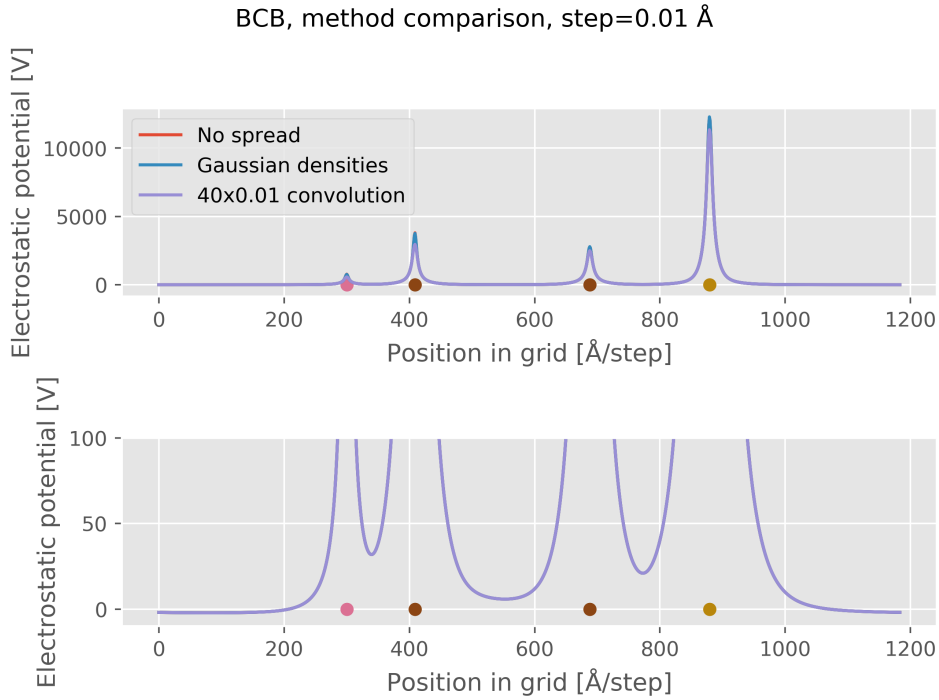


Figure A.3: Behaviour of different spreading methods with step size 0.01 Å. Only small differences can be seen between the methods at the positions of the nuclei. Between the nuclei and outside the molecule the lines go on top of each other.

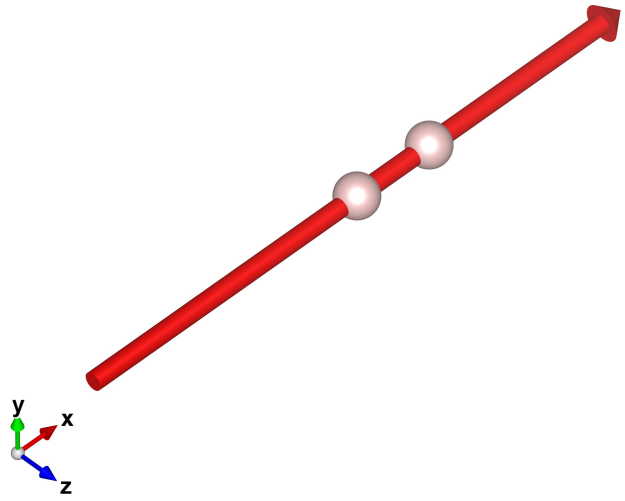
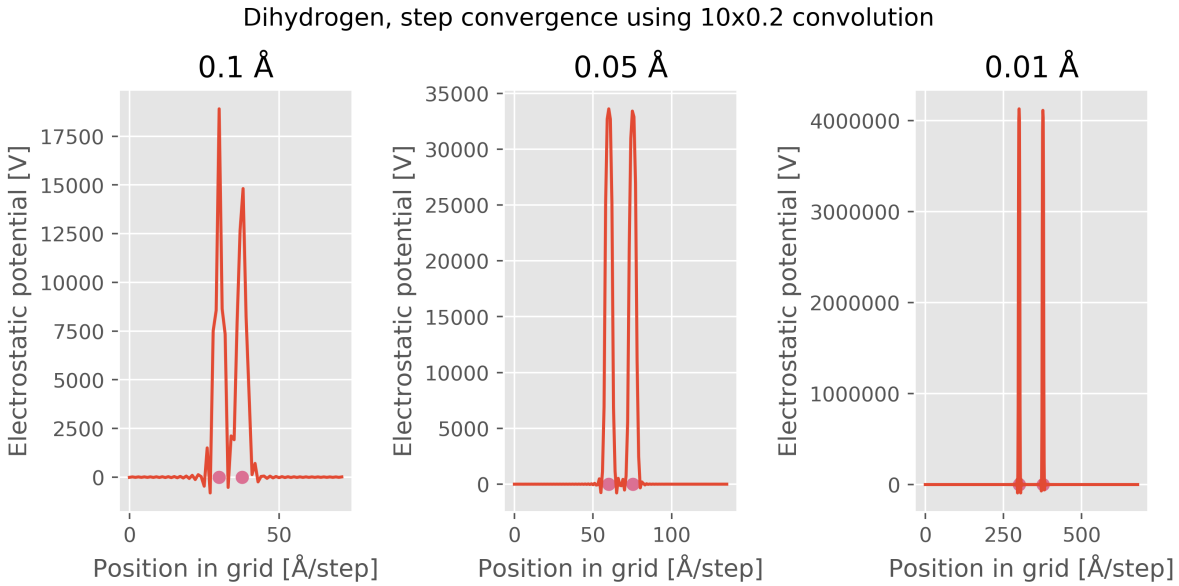
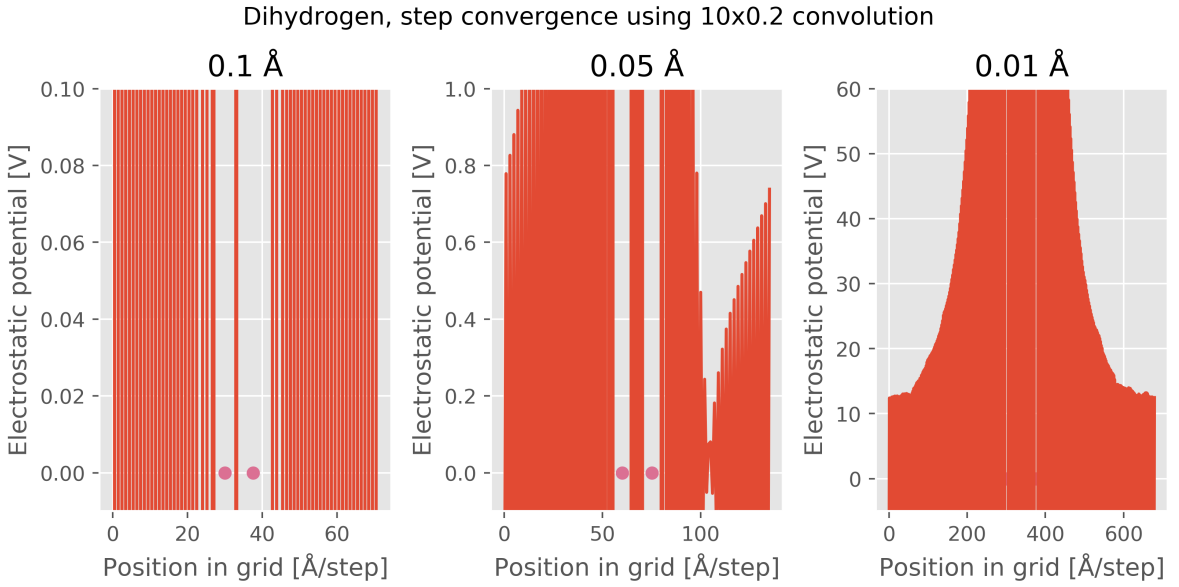


Figure A.4: Path used for the analysis of a dihydrogen molecule.

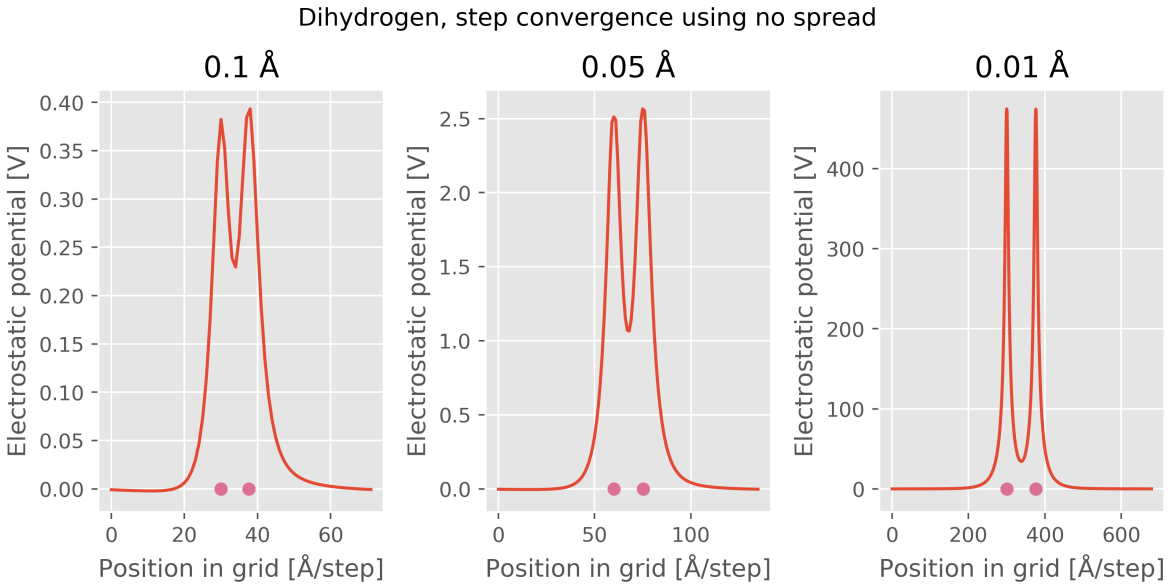


(a)

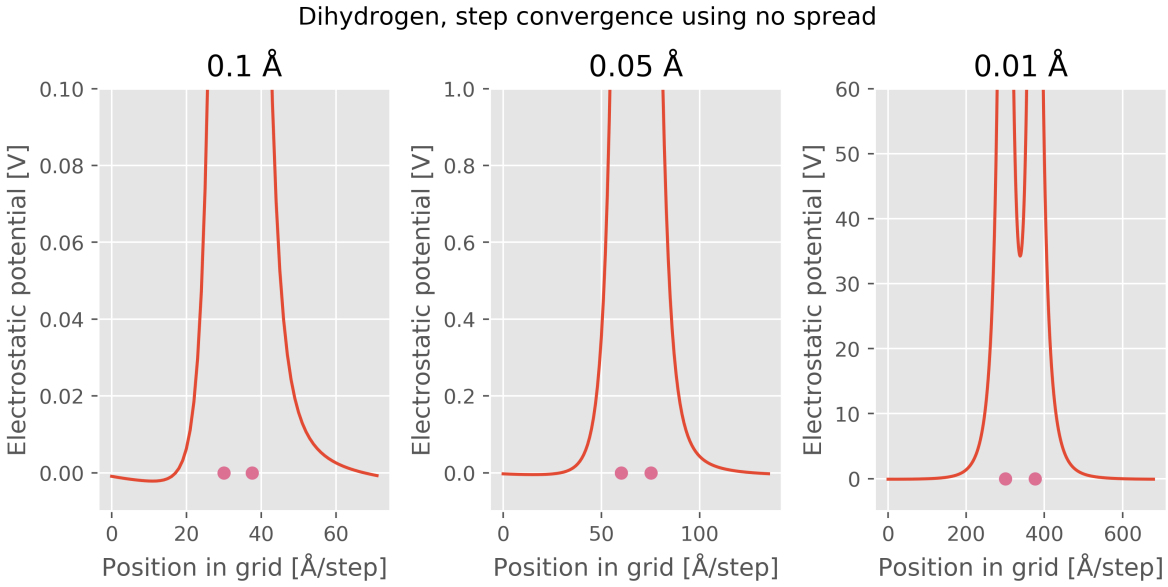


(b)

Figure A.5: Gibb's phenomenom appearing in analysis for a dihydrogen molecule with bad convolution constants. Zoomed in values shown in subfigure **(b)**

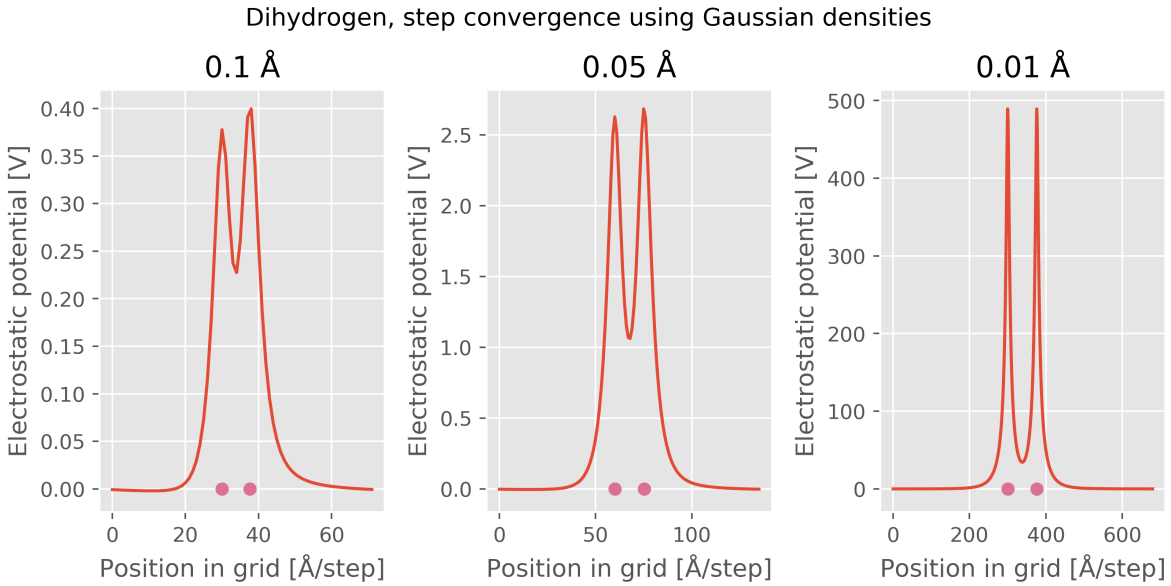


(a)

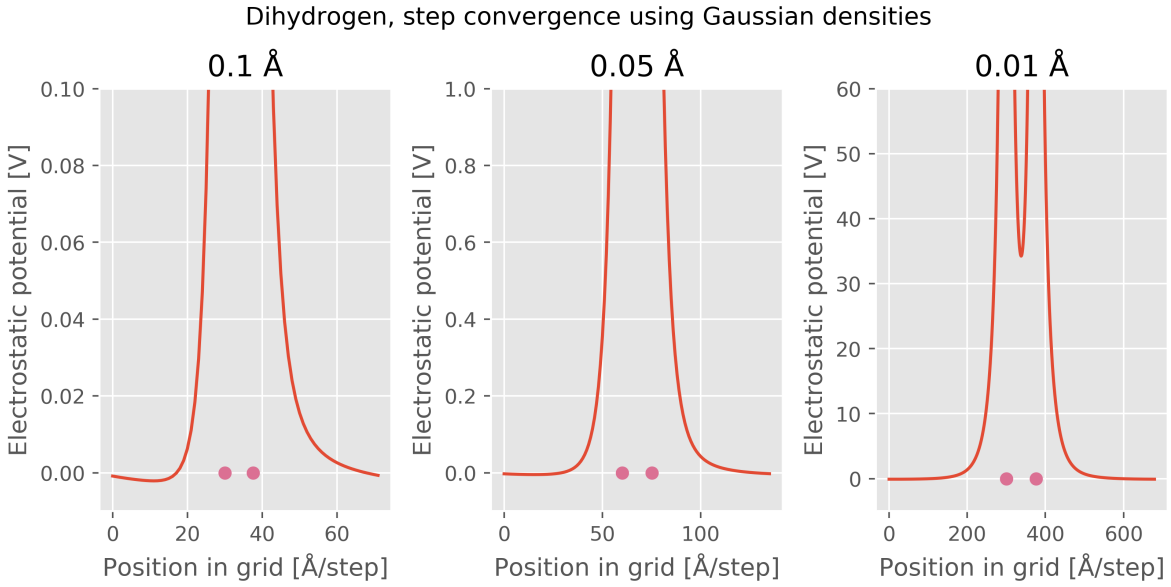


(b)

Figure A.6: Values for electrostatic potential when passing through the nuclei of a dihydrogen, with subfigure **(b)** having zoomed in y-axes. No spreading was used for the nuclear charges.



(a)



(b)

Figure A.7: Values for electrostatic potential when passing through the nuclei of a dihydrogen, with subfigure **(b)** having zoomed in y-axes. Gaussian densities were used for the nuclear charges.

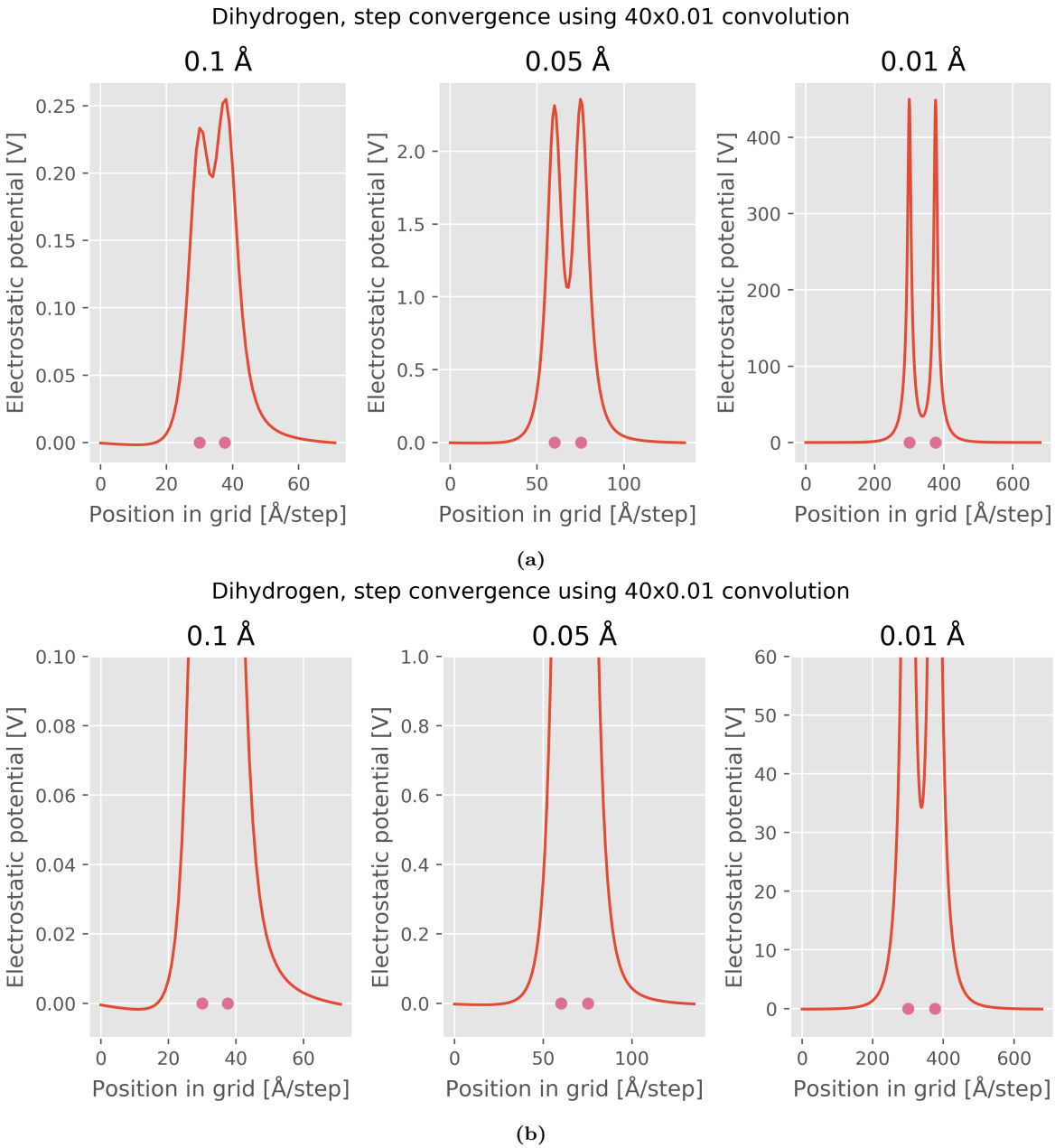


Figure A.8: Values for electrostatic potential when passing through the nuclei of a dihydrogen, with subfigure **(b)** having zoomed in y-axes. Convolving with constants $N_d = 40$ and $D = 0.01$ were used for the nuclear charges.

Appendix B. Code

Code produced for the project can be found at:
https://github.com/toicca/molecule-potential

Bibliography

[1] Niko Oinonen, Chen Xu, Benjamin Alldritt, Filippo Federici Canova, Fedor Urtev, Ondřej Krejčí, Juho Kannala, Peter Liljeroth, and Adam S. Foster. Electrostatic Discovery Atomic Force Microscopy, 2021.

[2] Wolfram Koch and Max C. Holthausen. *A Chemist's Guide to Density Functional Theory*. Wiley-VCH, Verlag GmbH, D-69469 Weinheim (Federal Republic of Germany), 2001.

[3] Max Born and J. Robert Oppenheimer. Zur Quantentheorie der Molekeln. (German) [on the quantum theory of molecules]. *Annalen der Physik*, 389(20):457–484, 1927.

[4] Igor Ying Zhang and Andreas Grüneis. Coupled cluster theory in materials science. *Frontiers in Materials*, 6:123, 2019.

[5] Trygve Helgaker, Poul Jørgensen, and Jeppe Olsen. *Molecular Electronic-Structure Theory*. John Wiley & Sons, Ltd, 2000.

[6] Rodney J. Bartlett and Monika Musiał. Coupled-cluster theory in quantum chemistry. *Rev. Mod. Phys.*, 79:291–352, Feb 2007.

[7] T. Daniel Crawford and Henry F. Schaefer III. *An Introduction to Coupled Cluster Theory for Computational Chemists*, pages 33–136. John Wiley & Sons, Ltd, 2000.

[8] Isaiah Shavitt and Rodney J. Bartlett. *Many-Body Methods in Chemistry and Physics: MBPT and Coupled-Cluster Theory*. Cambridge Molecular Science. Cambridge University Press, 2009.

[9] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136:B864–B871, Nov 1964.

[10] Thom H. Dunning. Gaussian basis sets for use in correlated molecular calculations. i. the atoms boron through neon and hydrogen. *The Journal of Chemical Physics*, 90(2):1007–1023, 1989.

[11] Anthony Croft and Robert Davison. *Engineering Mathematics, 5th Edition*. Pearson, 2017.

[12] Henri J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms*. Springer Series in Information Sciences. Springer, 1982.

[13] Nvidia. cufft library user's guide, `https://docs.nvidia.com/cuda/cufft/index.html`, (accessed: 11.09.2021).

[14] James Cooley and John Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[15] M. Frigo and S.G. Johnson. The design and implementation of fftw3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[16] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., USA, 1989.

[17] Nvidia. Cuda c++ programming guide, `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`, (accessed: 11.09.2021).

[18] John P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Books on Mathematics. Dover Publications, Mineola, NY, second edition, 2001.