A GPU accelerated and error-controlled solver for the unbounded Poisson equation in three dimensions

Lukas Exl *1,2

^{1}Fak. Mathematik, Univ. Wien, Oskar-Morgenstern-Platz 1, 1090, Vienna.
^{2}Inst. of Solid State Physics, TU Wien, Karlsplatz 13, 1040, Vienna.

November 14, 2018

Abstract. An efficient solver for the three dimensional free-space Poisson equation is presented. The underlying numerical method is based on finite Fourier series approximation. While the error of all involved approximations can be fully controlled, the overall computation error is driven by the convergence of the finite Fourier series of the density. For smooth and fast-decaying densities the proposed method will be spectral accurate. The method scales with $\mathcal{O}(N \log N)$ operations, where N is the total number of discretization points in the Cartesian grid. The majority of the computational costs come from fast Fourier transforms (FFT), which makes it ideal for GPU computation. Several numerical computations on CPU and GPU validate the method and show efficiency and convergence behavior. Tests are performed using the Vienna Scientific Cluster 3 (VSC3). A free MATLAB implementation for CPU and GPU is provided to the interested community.

Keywords: convolution via fast Fourier transform (FFT), GPU computing, free space Coulomb/ dipole-dipole potential, separable Gaussian-sum (GS) approximation

1 Introduction

The purpose of this paper is to provide the interested reader with a MATLAB implementation of an efficient and mathematically analyzed method [1] for solving the free-space/unbounded Poisson equation. More precisely, the method presented in this paper solves

$$-\Delta u(\mathbf{x}) = \rho(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^3, \quad \lim_{|\mathbf{x}| \to \infty} |u(\mathbf{x})| = 0, \tag{1}$$

via the well-known representation of the solution to (1) as the convolution of the density ρ with the free-space Green's function $U(\mathbf{x}) = \frac{1}{4\pi} \frac{1}{|\mathbf{x}|}$

$$u(\mathbf{x}) = (U * \rho)(\mathbf{x}) = \int_{\mathbb{R}^3} U(\mathbf{x} - \mathbf{y})\rho(\mathbf{y})d\mathbf{y}, \quad \mathbf{x} \in \mathbb{R}^3. \tag{2}$$

The problem (1) is fundamental in many fields of physics, e.g. quantum chemistry [2–7], particle physics [8, 9] or astrophysics [10]. Therefore, the provided implementation might serve as improvement of existing simulation codes that use the high-level computing environment MAT-LAB. However, the provided code could also be understood as an easily readable open source

---

*lukas.exl@univie.ac.at

prototype, ready for translation to other different programming languages.

In the following Sec. 2 the method is described mathematically, followed by a section about computational aspects and approximation errors (see Sec. 3). Sec. 4 describes the usage of the implementation by means of a test example. Validation of the implementation and tests for computational efficiency (see Sec. 5) show practical applicability. Test runs are performed on the Vienna Scientific Cluster 3 (VSC3) both on CPU nodes and Tesla GPU devices.

2 Method description

The method of this paper is in the class of Ewald type methods [3, 11, 12]. Those approaches split the singular convolution kernel U into a smooth long-range part U_s and a singular short-range correction U_c. The smooth part of the convolution can then be treated with help of the convolution theorem, i.e., $U * \rho = \mathcal{F}^{-1}(\mathcal{F}(U_s) \cdot \mathcal{F}(\rho))$. This is usually done on an equispaced Cartesian grid with the help of the quasi linearly scaling fast Fourier transform. Here, the smoothness usually leads to fast converging Fourier series, which make the discrete approximation accurate even on coarser grids. However, the correction U_c still contains a singularity but is also localized, hence, can be treated with a direct summation approach. Wile a direct evaluation of the convolution (1) would scale with $\mathcal{O}(N^2)$ on a Cartesian grid with a total number of N grid points, the original Ewald method [11] and parameter tuned variations of it scale with $\mathcal{O}(N^{3/2})$ operations. The method described here scales with $\mathcal{O}(N \log N)$ operations. This is achieved by FFT for both parts, the smooth convolution and the correction. The smooth kernel consists of a product of one-dimensional exponential functions (Gaussian-sum), which allows the usage of highly accurate one-dimensional adaptive quadrature for computation of the interaction kernel in Fourier space. Taylor expansion of the density in the near-zone allows to treat the correction by analytical integration, where involved derivatives are computed by FFT as well. The method is efficient and mathematically proven to yield full control over the maximum computation error [1].

We now give a brief description of the method, where we also emphasize novel aspects relevant to the implementation. A detailed mathematical description of the method including error analysis was recently published by the author [1]. We adapt it here for the case of general rectangular computational domains.

The computational box coincides with the domain of target points $\mathbf{x}$, where the potential u is computed. The smooth density ρ is assumed to vanish (up to double precision) outside the computational box, so it is expected to be fast decaying. Our method makes use of a finite Fourier series approximation of the density, which is assumed to be fast converging due to the smoothness and compact support of ρ. The analysis in [1] assumes the computational domain to be the unit square box $\mathbf{B}_1 := [-1, 1]^3$. To generalize the method's framework for the user's convenience, assume the density ρ to be compactly supported in the general rectangular box $\mathbf{B} := [a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$. We will first derive a 'standardized form' of the problem, which allows us to treat the key-approximations of the method independently of the concrete choice of the computational box. Let $\mathbf{c}$ be the center of $\mathbf{B}$ such that $\widetilde{\mathbf{B}} := \mathbf{B} - \mathbf{c}$ is a rectangular box centered at the origin. As a consequence of the compact support of the density, the convolution integral (2) is actually over the domain $\mathbf{B}$. Hence, we can write

$$u(\mathbf{x}) = \int_{\widetilde{\mathbf{B}}} U(\widetilde{\mathbf{x}} - \widetilde{\mathbf{y}}) \rho(\widetilde{\mathbf{y}} + \mathbf{c}) d\widetilde{\mathbf{y}}, \quad \mathbf{x} = \widetilde{\mathbf{x}} + \mathbf{c} \in \mathbf{B}. \tag{3}$$

Now we define $\lambda := \max_{q=1,2,3}\{\frac{b_q-a_q}{2}\}$ and $\mathbf{B}_{1,\lambda} := \frac{1}{\lambda}\widetilde{\mathbf{B}} \subseteq \mathbf{B}_1$. We get

$$u(\mathbf{x}) = \lambda^2 \int_{\mathbf{B}_{1,\lambda}} U(\mathbf{x}' - \mathbf{y}')\rho(\lambda\mathbf{y}' + \mathbf{c})d\mathbf{y}', \quad \mathbf{x} = \lambda\mathbf{x}' + \mathbf{c} \in \mathbf{B}. \tag{4}$$

Changing variables and extending the domain of integration to $2 \cdot \mathbf{B}_{1,\lambda} \supseteq \mathbf{B}_{1,\lambda} - \mathbf{x}'$, $\mathbf{x}' \in \mathbf{B}_{1,\lambda}$ leads to

$$u(\mathbf{x}) = \lambda^2 \int_{2\cdot\mathbf{B}_{1,\lambda}} U(\mathbf{y})\rho_{\lambda;\mathbf{c}}(\mathbf{x}' - \mathbf{y})d\mathbf{y}, \quad \mathbf{x} = \lambda\mathbf{x}' + \mathbf{c} \in \mathbf{B}, \tag{5}$$

where $\rho_{\lambda;\mathbf{c}}(\mathbf{x}) := \rho(\lambda\mathbf{x} + \mathbf{c})$ with support in $\mathbf{B}_{1,\lambda}$. The key idea of the method is to approximate the singular kernel $U(\mathbf{x}) = \frac{1}{4\pi}\frac{1}{|\mathbf{x}|}$ with a Gaussian-sum in a region contained in the integration domain $2 \cdot \mathbf{B}_{1,\lambda}$ but excluding a δ-ball around the origin (where U is singular). The latter step is compensated by a near zone correction. More precisely, for $\delta \in (0, \min_{q=1,2,3}\{\frac{b_q-a_q}{2\lambda}\})$ and $\mathbf{x} = \lambda\mathbf{x}' + \mathbf{c} \in \mathbf{B}$ we get

$$u(\mathbf{x}) \approx \lambda^2 \Big(\int_{2\cdot\mathbf{B}_{1,\lambda}} U_{GS}(\mathbf{y})\rho_{\lambda;\mathbf{c}}(\mathbf{x}' - \mathbf{y})d\mathbf{y} + \int_{\mathbf{B}_\delta} (U - U_{GS})(\mathbf{y})\rho_{\lambda;\mathbf{c}}(\mathbf{x}' - \mathbf{y})d\mathbf{y} \Big) =: \lambda^2\big(I_1(\mathbf{x}) + I_\delta(\mathbf{x})\big), \tag{6}$$

where $U_{GS}(\mathbf{y}) = \sum_{j=0}^{S} w_j e^{-\tau_j^2|\mathbf{y}|^2} = \sum_{j=0}^{S} w_j \prod_{q=1}^{3} e^{-\tau_j^2 y_q^2} \approx U(\mathbf{y})$, $|\mathbf{y}| \in [\delta, 2]$ is a Gaussian-sum (GS) approximation realized by sinc-quadrature [1, 13, 14]. The integrand in $I_1(\mathbf{x})$ is smooth and its convolution kernel is separable (product of 1d functions). Hence, it can be treated efficiently by an Fourier based approach. More precisely, it is computed by the inverse Fourier transform of the product of the Fourier transform of the density with the G-tensor

$$G_{\mathbf{k}} = \sum_{j=0}^{S} w_j G_{\mathbf{k}}^j \quad \text{with } G_{\mathbf{k}}^j = G_{k_1 k_2 k_3}^l = \prod_{q=1}^{3} \int_0^2 2l_q \, e^{-(\tau_j l_q)^2 y_q^2} \cos(\frac{\pi}{2}k_q y_q) \, dy_q, \tag{7}$$

where $\mathbf{B}_{1,\lambda} = [-l_1, l_1] \times [-l_2, l_2] \times [-l_3, l_3]$. The G-tensor can be computed accurately by one-dimensional adaptive Gauss-Kronrod quadrature in a setup phase. The two Fourier transforms in the (run-time) computation of I_1 are efficiently implemented via the FFT with zero-padding, which increases the effort by a factor of eight.

The correction integral I_δ is calculated by inserting the third order Taylor polynomial of the shifted density $\rho_{\lambda;\mathbf{c};\mathbf{x}'}(\mathbf{y}) := \rho_{\lambda;\mathbf{c}}(\mathbf{x}' - \mathbf{y})$ around $\mathbf{0}$, followed by analytical integration in spherical coordinates. The contributions of odd derivatives in the Taylor expansion and the off-diagonal elements of the Hessian cancel out. The remaining derivatives are computed from the finite Fourier series of the density, which makes it a scalar multiplication. This step is realized by using forward and backward FFT.

3   Computational aspects and approximation errors

For the concrete computation the computational box is discretized equidistantly with $N := n_1 n_2 n_3$ Cartesian grid points, i.e., the q-th principal direction is discretized equidistantly with n_q points. The solver's setup phase consists mainly of the precomputation of the G-tensor. The two Fourier transforms (one forward, one backward) in the computation of I_1 in (6) are of size $8N$ and scale with $\mathcal{O}(8N \log 8N)$ operations utilizing the FFT. The evaluation of the near zone correction makes use of two FFTs (one forward, one backward) of size N and therefore scales with $\mathcal{O}(N \log N)$ operations. Other operations (multiplications and additions) contribute with

linear scaling $\mathcal{O}(N)$.

Besides the error coming from the finite Fourier series approximation of the density, the (maximum-) error of the convolution method is (i) in the computation of I_1 due to the Gaussian-sum approximation in $[\delta, 2]$ and (ii) in the computation of the correction I_δ due to the Taylor expansion of the density. For fixed δ the error (i) is controlled by a parameter $\epsilon > 0$ in the order of around machine precision, while the error (ii) amounts to δ^6. The overall maximum-error of the involved approximations is therefore in the order of $\max\{\epsilon, \delta^6\}$. In practice, δ will be around $0.005 - 0.001$, hence, yielding an overall maximum-error of the involved approximations of around ϵ. Thus, the overall computation error can be expected to be determined by the convergence of the finite Fourier series of the density (spectral accuracy). It is also known from the error analysis in [1] that the approximations of the method without the correction I_δ yield a maximum-error in the order of $\max\{\delta^2, \epsilon\} \sim 10^{-6}$. If the error coming from the finite Fourier series lies above this threshold, the correct I_δ will lead to no improvement. For coarse grids the error from the Fourier series can be expected to exceed the threshold, such that δ does not have to be chosen too small. This reflects in the computation time of the setup phase, since a larger choice of δ leads to a smaller number S of terms in the precomputation of the G-tensor. The heuristic choice of $\delta := \min \frac{b_q - a_q}{50 n_q} \in (0, \min_{q=1,2,3}\{\frac{b_q - a_q}{2\lambda}\})$ for $\lambda \leq 25 \min_q n_q$ depends on the discretization size according to the just mentioned considerations. However, we take $\delta \geq 10^{-3}$ as a minimum threshold, since the I_δ-correction yields accuracy in the order of δ^6.

4 Usage of the solver

Usage of the solver is simple, see Listing 1. The user defines the computational box $\mathbf{B}$ and its discretization by uniformly discretized edges. Next the GPU flag is set and the setup of the solver is accomplished. In Listing 1 a Gaussian test density with compact support in $\mathbf{B}$ is chosen which is sampled on the discretized computational box. The actual computation is performed by the solver's `solve` method. Afterwards the solver could be reused without renewed setup, e.g. in large simulations where the potential has to be computed several times on the same geometry. In Listing 1 the error computation is demonstrated as well, which is only possible for the analytically given test density.

Listing 1: Usage of the `GSPoisson3d` solver.

```
%================================================================
% Setup geometry
%================================================================

% Define the computational box
x_min = -2.0;
y_min = -2.0;
z_min = -2.0;

x_max = +2.0;
y_max = +2.0;
z_max = +2.0;

% Number of discretization points for each principal direction
Jx = 2^6;
Jy = 2^6;
Jz = 2^6;

% Uniformly discretized axes
x = linspace(x_min, x_max, Jx);
```

```matlab
y = linspace(y_min, y_max, Jy);
z = linspace(z_min, z_max, Jz);

%================================================================
% Setup solver
%================================================================

% GPU flag
use_gpu = false; % true;

% setup solver
solver = GSPoisson3d(x, y, z, use_gpu);

%================================================================
% Density
%================================================================

% example density
[f, u_ref] = problems.gaussian(x, y, z, 0.2);

if use_gpu == true
    f     = gpuArray(f);
    u_ref = gpuArray(u_ref);
end

%================================================================
% Actual computation
%================================================================

tic;
u = solver.solve(f);
time = toc;

% Error
E = max( abs( u(:) - u_ref(:) ) ) / max( abs( u_ref(:) ) );

if use_gpu == true
    E = gather(E);
end

fprintf('Maximum error: %.4e\n', E)
fprintf('Time: %.3f (s)\n',time)
```

5   Numerical validation

We test our solver for different choices of the density and give maximum relative errors E according to

$$E := \frac{\|u - u_{\vec{h}}\|_{l^\infty}}{\|u\|_{l^\infty}} = \frac{\max_{\mathbf{x} \in \mathcal{T}_h} |u(\mathbf{x}) - u_{\vec{h}}(\mathbf{x})|}{\max_{\mathbf{x} \in \mathcal{T}_h} |u(\mathbf{x})|}, \tag{8}$$

where $\mathcal{T}_h$ is the rectangular computational domain discretized uniformly in each direction with mesh sizes $\vec{h} = (h_x, h_y, h_z)^T$. Errors and computation times are compared for the CPU and the GPU case. In the following we denote our solver with GSPoisson3d solver. The computations were submitted jobs on the Vienna Scientific Cluster 3 (VSC3) which consists of nodes with Intel Xeon E5-2650v2 2.6GHz processors and Tesla K20m GPU devices. To accurately give the timings we measure the average times of 100 computations.

5.1 Gaussian source

First we test with the Gaussian density

$$\rho(\mathbf{x}) = \frac{1}{(2\pi)^{3/2}\sigma^3} e^{-|\mathbf{x}-\mathbf{c}|^2/(2\sigma^2)}, \tag{9}$$

where $\mathbf{c} \in \mathbb{R}^3$ is the center of the computational box. The exact solution is known to be

$$u^*(\mathbf{x}) = \frac{1}{4\pi|\mathbf{x}-\mathbf{c}|}\text{Erf}\Big(\frac{|\mathbf{x}-\mathbf{c}|}{\sqrt{2}\sigma}\Big). \tag{10}$$

We vary the shape parameter σ in our tests and compare errors and computation times on CPU and GPU, see Tab. 1 for the computational domain $\mathbf{B} = [-2, 2]^3$.

Table 1: Errors and timings for Gaussian density (9) in $[-2, 2]^3$. Errors E, times t_{cpu} and t_{gpu} on CPU and GPU respectively.

σ	N	E	t_{cpu}	t_{gpu}
0.20	16^3	1.659E-03	6.80E-03	1.20E-02
0.20	32^3	4.154E-09	1.91E-02	6.27E-03
0.20	64^3	6.197E-16	1.17E-01	1.17E-02
0.20	128^3	1.052E-15	7.95E-01	5.29E-02
0.15	16^3	2.986E-02	4.37E-03	5.44E-03
0.15	32^3	2.937E-06	1.80E-02	6.26E-03
0.15	64^3	9.386E-16	1.17E-01	1.20E-02
0.15	128^3	1.187E-15	8.66E-01	5.26E-02
0.10	16^3	3.802E-01	4.34E-03	6.39E-03
0.10	32^3	1.129E-03	1.91E-02	6.26E-03
0.10	64^3	2.624E-09	1.18E-01	1.10E-02
0.10	128^3	1.593E-15	8.22E-01	5.26E-02

Fig. 1 shows the convergence of the method for example (9) with $\sigma = 0.05$ in $\mathbf{B} = [-2, 2]^3$.

This examples show that the `GSPoisson3d` solver converges up to exponentially fast. Fig. 2 compares computation times for CPU and GPU for the computations associated with those of Fig. 1. Computation times show the $N \log N$ scaling. The GPU acceleration yields a speed up of up to 10 in the case of larger N.

Finally, Tab. 2 shows results for the rectangular computational domain $\mathbf{B} = [-3, 2] \times [-2, 3.5] \times [-1, 5]$ and $\sigma = 0.2$.

5.2 Superposition of Gaussian sources

We test with the Gaussian density

$$\rho(\mathbf{x}) = \frac{1}{2}\Big(\frac{1}{(2\pi)^{3/2}\sigma_1^3} e^{-|\mathbf{x}-(\mathbf{c}+\mathbf{d})|^2/(2\sigma_1^2)} + \frac{1}{(2\pi)^{3/2}\sigma_2^3} e^{-|\mathbf{x}-(\mathbf{c}-\mathbf{d})|^2/(2\sigma_2^2)}\Big), \tag{11}$$
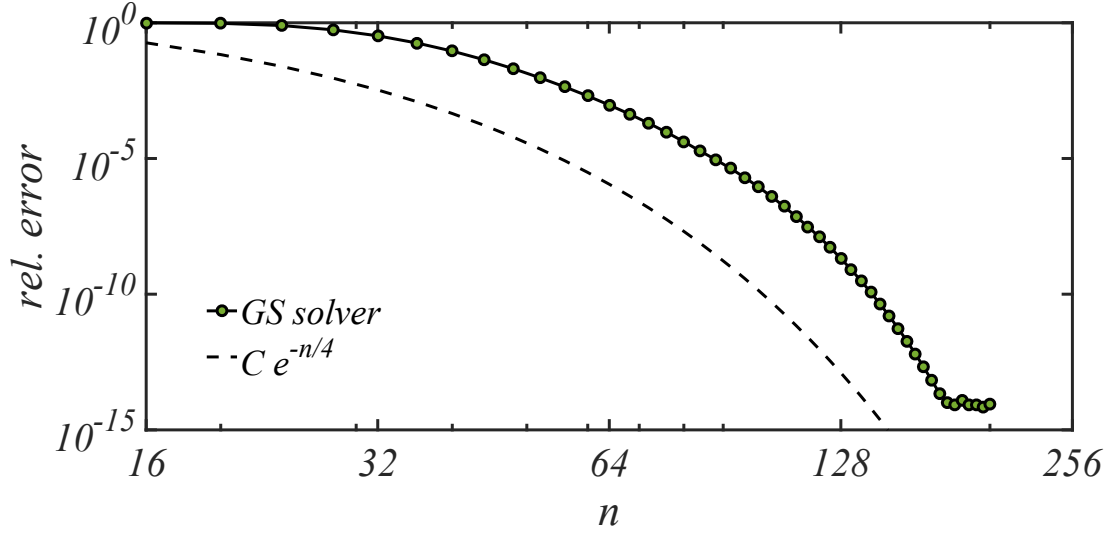
Figure 1: Errors for the density in (9) in $\mathbf{B} = [-2, 2]^3$ with $\sigma = 0.05$.
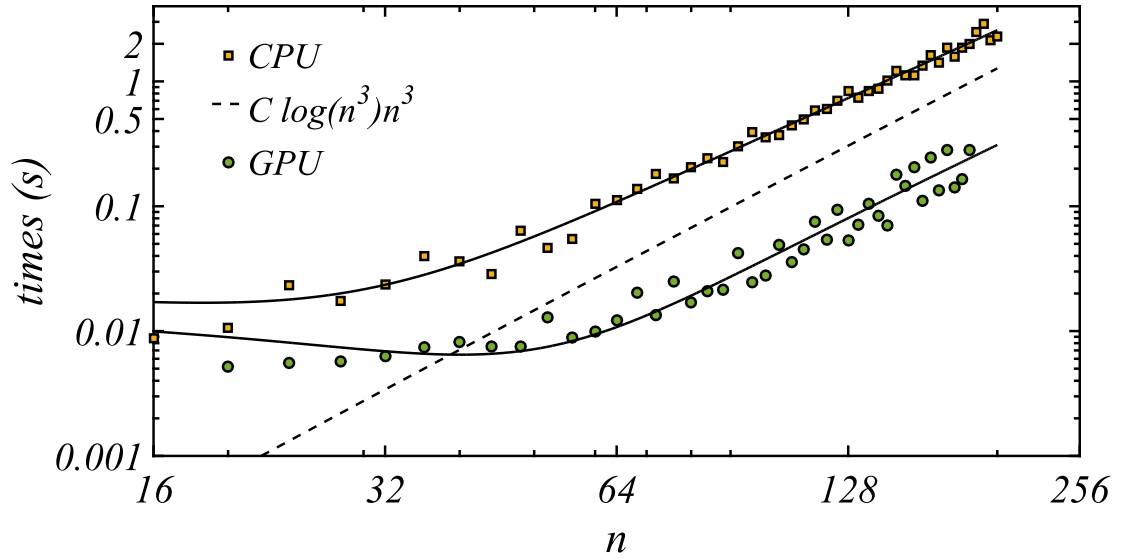

Figure 2: Computation times for CPU and GPU on the Vienna Scientific Cluster 3 (VSC3) for the density in (9) in $\mathbf{B} = [-2, 2]^3$ with $\sigma = 0.05$.

Table 2: Errors and timings for Gaussian density (9) in $[-3, 2] \times [-2, 3.5] \times [-1, 5]$ and $\sigma = 0.2$. Errors E, times t_{cpu} and t_{gpu} on CPU and GPU respectively.

N	E	t_{cpu}	t_{gpu}
16^3	4.417E-02	6.65E-03	1.02e-02
32^3	1.857E-05	1.89E-02	7.58e-03
64^3	3.126E-12	1.29E-01	1.21e-02
128^3	1.643E-12	8.18E-01	5.28e-02

where $\mathbf{c}, \mathbf{d} \in \mathbb{R}^3$ is the center of the computational box and a shift, respectively. The exact solution is

$$u^*(\mathbf{x}) = \frac{1}{2}\left(\frac{1}{4\pi|\mathbf{x} - (\mathbf{c}+\mathbf{d})|} \text{Erf}\left(\frac{|\mathbf{x} - (\mathbf{c}+\mathbf{d})|}{\sqrt{2}\sigma_1} \right) + \frac{1}{4\pi|\mathbf{x} - (\mathbf{c}-\mathbf{d})|} \text{Erf}\left(\frac{|\mathbf{x} - (\mathbf{c}-\mathbf{d})|}{\sqrt{2}\sigma_2} \right) \right). \quad (12)$$

We compare errors and computation times on CPU and GPU and give the results in Tab. 3.

Table 3: Errors and timings for Gaussian density (11) in $[-2, 2]^3$ with $\mathbf{d} = (0.1, -0.05, 0.05)^T$ and $\sigma_1 = 0.2$ and $\sigma_2 = 0.1$. Errors E, times t_{cpu} and t_{gpu} on CPU and GPU respectively.

N	E	t_{cpu}	t_{gpu}
16^3	5.663E-02	1.14E-02	2.12e-02
32^3	1.533E-03	2.26E-02	6.65e-03
64^3	5.920E-09	1.30E-01	1.17e-02
128^3	1.246E-15	8.72E-01	5.26e-02

5.3 Bump function

Next we consider the following bump function as density $(d, R > 0)$

$$\rho_{R,c}(\mathbf{x}) = \begin{cases} 2dR^2 \frac{3R^4 - 2R^2|\mathbf{x}-\mathbf{c}|^2 - |\mathbf{x}-\mathbf{c}|^4 - dR^2|\mathbf{x}-\mathbf{c}|^2}{(R^2 - |\mathbf{x}-\mathbf{c}|^2)^4} e^{-dR^2(R+|\mathbf{x}-\mathbf{c}|)/(R-|\mathbf{x}-\mathbf{c}|)}, & |\mathbf{x} - \mathbf{c}| < R \\ 0, & |\mathbf{x} - \mathbf{c}| \geq R, \end{cases} \quad (13)$$

where the exact solution is given as

$$u^*(\mathbf{x}) = \begin{cases} e^{\frac{-d}{1-|\mathbf{x}-\mathbf{c}|^2/R^2}} & |\mathbf{x} - \mathbf{c}| < R \\ 0, & |\mathbf{x} - \mathbf{c}| \geq R. \end{cases} \quad (14)$$

Tab. 4 shows the results in the rectangular domain $[-3, 1] \times [-2, 3] \times [-2, 4]$ with $d = 10$ and $R = 2$.

5.4 Anisotropic Gaussian

We test for the anisotropic density $(\mathbf{c} = (c_x, c_y, c_z)^T, \ \sigma_x, \sigma_y, \sigma_z > 0)$

$$\rho(\mathbf{x}) = -\left(\frac{4(x-c_x)^2}{\sigma_x^4} + \frac{4(y-c_y)^2}{\sigma_y^4} + \frac{4(z-c_z)^2}{\sigma_z^4} - \frac{2}{\sigma_x^2} - \frac{2}{\sigma_y^2} - \frac{2}{\sigma_z^2} \right) e^{-(x-c_x)^2/\sigma_x^2 - (y-c_y)^2/\sigma_y^2 - (z-c_z)^2/\sigma_z^2}, \quad (15)$$

8

Table 4: Errors and timings for the density in (13) in $[-3, 1] \times [-2, 3] \times [-2, 4]$ with $d = 10$ and $R = 2$. Errors E, times $t_{\mathtt{cpu}}$ and $t_{\mathtt{gpu}}$ on CPU and GPU respectively.

N	E	$t_{\mathtt{cpu}}$	$t_{\mathtt{gpu}}$
16^3	2.070E-03	6.64E-03	1.05E-02
32^3	3.928E-06	2.26E-02	7.76E-03
64^3	9.264E-10	1.19E-01	1.18E-02
128^3	4.973E-13	7.95E-01	5.26E-02

which is produced by Eqn. (1) and the prescribed exact solution

$$u^*(\mathbf{x}) = e^{-(x-c_x)^2/\sigma_x^2 - (y-c_y)^2/\sigma_y^2 - (z-c_z)^2/\sigma_z^2}. \tag{16}$$

Tab. 5 shows the results for $\boldsymbol{\sigma} = (0.30, 0.20, 0.28)^T$ on $\mathbf{B} = [-2, 2]^3$.

Table 5: Errors and timings for the density in (15) in $[-2, 2]^3$ with $\boldsymbol{\sigma} = (0.30, 0.20, 0.28)^T$. Errors E, times $t_{\mathtt{cpu}}$ and $t_{\mathtt{gpu}}$ on CPU and GPU respectively.

N	E	$t_{\mathtt{cpu}}$	$t_{\mathtt{gpu}}$
16^3	4.208E-01	6.51E-03	1.73E-02
32^3	1.627E-04	2.07E-02	8.54E-03
64^3	1.466E-13	1.24E-01	1.17E-02
128^3	1.349E-15	8.01E-01	5.29E-02

Table 6: Errors and timings for the density in (15) in $\mathbf{B} = [-2, 2] \times [-4, 4] \times [-6, 6]$ with $\boldsymbol{\sigma} = (0.10, 0.20, 0.3)^T$. Errors E, times $t_{\mathtt{cpu}}$ and $t_{\mathtt{gpu}}$ on CPU and GPU respectively.

N	E	$t_{\mathtt{cpu}}$	$t_{\mathtt{gpu}}$
16^3	1.011e+02	6.51E-03	1.73E-02
32^3	9.529e-01	2.07E-02	7.54E-03
64^3	1.544e-04	1.24E-01	1.16E-02
128^3	7.839e-09	8.01E-01	5.29E-02
162^3	8.105e-09	1.22E+00	1.08E-01

The next experiment takes $\boldsymbol{\sigma} = (0.10, 0.20, 0.30)^T$ and a rectangular domain $\mathbf{B} = [-2, 2] \times [-4, 4] \times [-6, 6]$, which is adjusted to the $\sigma-$values. Tab. 6 shows the results for different mesh sizes. Convergence stagnates here at an error level of around 1e-9 most likely due to significant loss of digits in the computation of the G-tensor. However, large aspect ratios are known to be difficult cases. In fact, the next example shows that for flat domains $\mathbf{B} = [-2, 2] \times [-2L, 2L]^2$ for $L = 2, 4, 8, 16$ and adjusted $\boldsymbol{\sigma} = (0.20, 0.20\,L, 0.20\,L)^T$ the error stagnates and convergence gets worse for increasing L, compare with Fig. 3. Tests on prolongated domains $\mathbf{B} = [-2, 2]^2 \times [-2L, 2L]$ show qualitatively comparable results. However, already relatively coarse discretizations yield still acceptable error levels if the rectangular domain is not too flat or prolongated.
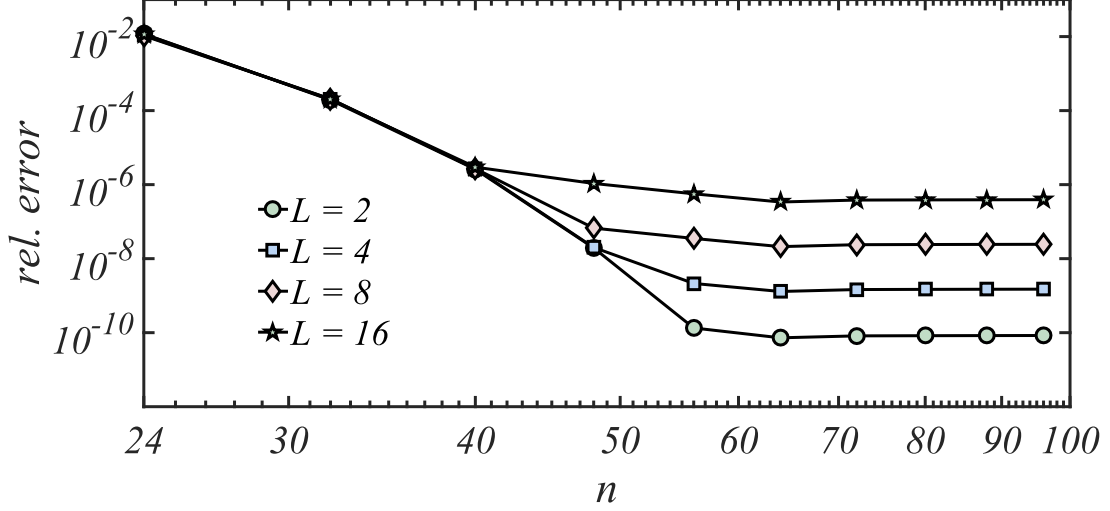
Figure 3: Errors for the density in (15) in $\mathbf{B} = [-2, 2] \times [-2L, 2L]^2$ for $L = 2, 4, 8, 16$ with $\boldsymbol{\sigma} = (0.20, 0.20\,L, 0.20\,L)^T$.

5.5   Oscillating density

We now test for the oscillating density $(\sigma, \omega > 0)$

$$\rho(\mathbf{x}) = e^{-|\mathbf{x}-\mathbf{c}|^2/\sigma^2}\left(6\omega\sin(\omega|\mathbf{x}-\mathbf{c}|^2) - 4\omega^2|\mathbf{x}-\mathbf{c}|^2\cos(\omega|\mathbf{x}-\mathbf{c}|^2)\right) \tag{17}$$

$$-\left(\frac{4|\mathbf{x}-\mathbf{c}|^2}{\sigma^4} - \frac{6}{\sigma^2}\right)e^{-|\mathbf{x}-\mathbf{c}|^2/\sigma^2}\cos(\omega\,|\mathbf{x}-\mathbf{c}|^2) - \frac{8\omega|\mathbf{x}-\mathbf{c}|^2}{\sigma^2}e^{-|\mathbf{x}-\mathbf{c}|^2/\sigma^2}\sin(\omega\,|\mathbf{x}-\mathbf{c}|^2) \tag{18}$$

which is produced by Eqn. 1 and the prescribed exact solution

$$u^*(\mathbf{x}) = e^{-|\mathbf{x}-\mathbf{c}|^2/\sigma^2}\cos(\omega\,|\mathbf{x}-\mathbf{c}|^2). \tag{19}$$

Tab. 7 shows the results for $\sigma = 0.30$ and $\omega = 20$ on $\mathbf{B} = [-2, 2]^3$.

Table 7: Errors and timings for the density in (17) in $[-2, 2]^3$ with $\sigma = 0.30$ and $\omega = 20$. Errors E, times $t_{\mathbf{cpu}}$ and $t_{\mathbf{gpu}}$ on CPU and GPU respectively.

N	E	$t_{\mathbf{cpu}}$	$t_{\mathbf{gpu}}$
16^3	6.179E+00	6.76E-03	7.93E-03
32^3	7.921E-03	1.85E-02	7.53E-03
64^3	3.631E-08	1.30E-01	1.17E-02
128^3	2.127E-15	8.24E-01	5.26E-02

As expected, the `GSPoisson3d` solver converges spectrally accurate for the oscillating density.

6   Conclusions

A solver for the solution of the free-space Poisson problem in three dimensions was presented and implemented in MATLAB for CPU and GPU usage. The method is spectral accurate and quasi

linearly scaling. The computational domain can be a general rectangular box, where numerical experiments indicate acceptable error levels for moderately flat or prolongated domains and anisotropic densities. The main computational tasks of the algorithm come from (zero-padded) FFTs. However, these operations are shown to be ideal for GPU acceleration, leading to a speedup factor compared to CPU of about 10 for the Tesla GPU on the Vienna Scientific Cluster 3 (VSC3). The proposed approach and provided MATLAB implementation [1] is shown to be practically useful in terms of accuracy and efficiency. However, the code could also be understood as an easily readable prototype for translation to different programming languages.

Acknowledgments

References

[1] L Exl, N J Mauser, and Y Zhang. Accurate and efficient computation of nonlocal potentials based on Gaussian-sum approximation. *J.Comput.Phys.*, 327:629–642, 2016.

[2] A R Leach. *Molecular modelling: principles and applications*. Pearson education, 2001.

[3] G J Martyna and M E Tuckerman. A reciprocal space based method for treating long range interactions in ab initio and force-field-based calculations in clusters. *J. Chem. Phys*, 110(6):2810–2821, 1999.

[4] L Genovese, T Deutsch, A Neelov, S Goedecker, and G Beylkin. Efficient solution of Poisson's equation with free boundary conditions. *J. Chem. Phys*, 125(7):074105, 2006.

[5] L Genovese, T Deutsch, and S Goedecker. Efficient and accurate three-dimensional Poisson solver for surface problems. *J. Chem. Phys*, 127(5):054704, 2007.

[6] L Füsti-Molnar and P Pulay. Accurate molecular integrals and energies using combined plane wave and gaussian basis sets in molecular electronic structure theory. *The Journal of chemical physics*, 116(18):7795–7805, 2002.

[7] W Bao, Y Cai, and H Wang. Efficient numerical methods for computing ground states and dynamics of dipolar bose–einstein condensates. *Journal of Computational Physics*, 229 (20):7874–7892, 2010.

[8] A Arnold and C Holm. Efficient methods to compute long-range interactions for soft matter systems. In *Advanced computer simulation approaches for soft matter sciences II*, pages 59–109. Springer, 2005.

[9] M M Hejlesen and J H Walther. A multiresolution method for solving the poisson equation using high order regularization. *Journal of Computational Physics*, 326:188–196, 2016.

[10] R D Budiardja and C Y Cardall. Parallel FFT-based Poisson solver for isolated three-dimensional systems. *Computer Physics Communications*, 182(10):2265–2275, 2011.

[11] P P Ewald. Die Berechnung optischer und elektrostatischer Gitterpotentiale. *Annalen der Physik*, 369(3):253–287, 1921.

---

[1]Available on the author's webpage.

[12] D M Heyes. Electrostatic potentials and fields in infinite point charge lattices. *The Journal of Chemical Physics*, 74(3):1924–1929, 1981.

[13] W Hackbusch and B N Khoromskij. Low-rank Kronecker-product approximation to multi-dimensional nonlocal operators. Part i. Separable approximation of multi-variate functions. *Computing*, 76(3-4):177–202, 2006.

[14] L Exl, C Abert, N J Mauser, T Schrefl, H P Stimming, and D Suess. FFT-based Kronecker product approximation to micromagnetic long-range interactions. *Math. Mod. Meth. Appl. S.*, 24(09):1877–1901, 2014.