POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

A Scalable Solver for the Linearized Poisson-Boltzmann Equation on Cartesian Grids with Hierarchical Local Refinement

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Martina Politi, Student ID 953448

Advisor:
Prof. Carlo de Falco

Co-advisors:
Dr. Walter Rocchia
Dr. Sergio Decherchi

Academic year:
2020-2021

Abstract: The understanding of the physical/chemical properties of molecules in aqueous solution is required to accurately describe the strength and nature of electrostatic interactions that play a central role in a variety of biological processes. The use of continuum solvent models such as the the Poisson–Boltzmann Equation (PBE) for modeling such interactions in biomolecular systems of ever growing complexity, heavily relies on the availability on accurate and efficient numerical methods and on the ability of their implementation to properly scale on modern highly parallel computer architectures. In this context, the present thesis presents the design and scalable implementation of `easy_pbe` a scalable numerical solver code for the *linearized* PBE. The solver is based on a Finite Element discretization of PBE on hierachically refined cartesian grids and allows to choose between various possible approaches for the description of the molecular surface. After introducing the problem formulation, and its non-dimensional form, we describe in particular detail some important aspects of the implemented solver such as the approach used to deal with singularities caused by point sources in the model, then we present a set of meaningful test cases to validate the code accuracy and its parallel scalability.

Key-words: Molecular electrostatics, Poisson–Boltzmann Equation, Molecular Surface, Oct-tree grid, MPI parallelization.

1 Introduction

This thesis deals with numerical methods and High Performance Computing (HPC) Techniques for the computation of the electrostatic potential at the surface of complex molecules (typically proteins) in aqueous solutions. The mathematical modeling tool we choose to use for this purpose is the Poisson Boltzmann Equation (PBE), which, in mathematical terms, is a boundary value problem for a semi-linear elliptic operator with discontinuous coefficient and point sources. In particular, we present the design and scalable implementation of `easy_pbe` [1] a numerical solver code for the *linearized* PBE. The understanding of the physical/chemical properties of molecules in aqueous solution is required to accurately describe the strength and nature of electrostatic interactions that play a central role in a variety of biological processes. The introduction of such model dates back almost exactly one century ago to the seminal work of Born [10], but its relevance for chemical and biological applications, *e.g.* for drug discovery, still holds [18], furthermore, when applied to large and complex molecules, such as, *e.g.* the SARS-CoV-2 spike protein [20], the solution of the PBE is a challenging benchmark for state–of–the–art HPC techniques. Among the many popular open implementations of PBE solvers we note, in particular, APBS [16]

and Delphi [23, 24, 26, 32], the former implements both a Finite Element (FE) discretization method on adaptive, conforming, simplicial meshes [5, 17] (more accurate, less efficient) and a Finite Differences (FD) scheme on tensor product cartesian grids, while the latter strongly relies on the benefits of FD schemes on cartesian grids in terms of memory efficiency and scalability. Here we focus on the performance and scalability assessment of numerical methods for the PBE based on hierarchically refined cartesian Oct-tree grids [2, 3], which is a topic that received a growing interest in the research community in recent years [11, 20, 25].
The thesis is organized as follows. In Section 2 we introduce the problem statement for the PBE and for its linearized version. In Section 2.1 a suitable scaling procedure is presented to reduce the PBE in its non-dimensional form, while in Section 2.2 two approaches used to deal with singularities caused by point sources are presented. One aspect in the modeling of protein electrostatics by means of the PBE that has been shown to play an important role both in terms of accuracy of the results and of computational efficiency, is that of the geometrical description of molecular surfaces [13, 35]; the comparison of different approaches for the surface definition is the subject of Section 3. In Section 4 we described the Oct-tree grids, focusing on their benefits in terms of efficient parallel partitioning, refinement, coarsening and balancing. The implementation aspects are described in Section 5. Finally, in Section 7, a set of numerical tests of growing complexity are presented in order to thoroughly validate the numerical algorithm and its implementation, while Section 8 contains the conclusions. In the Appendix a minimal user reference for `easy_pbe` is included for sake of convenience.

2 The Poisson–Boltzmann Equation

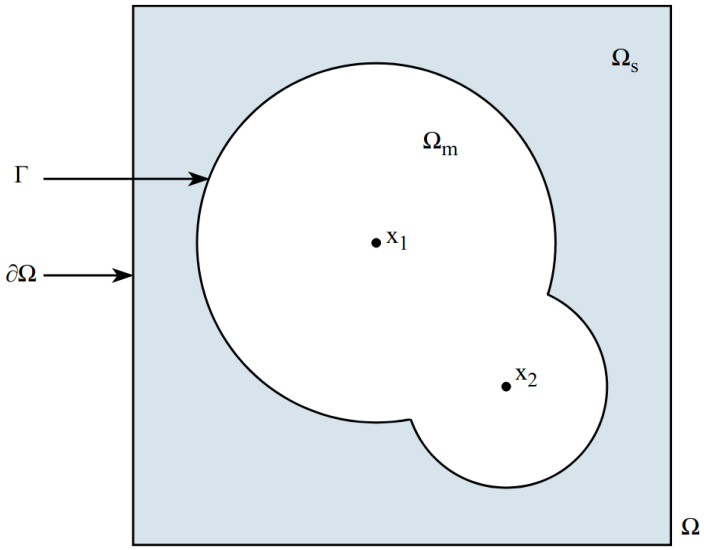


Figure 1: Schematic representation of the problem domain.

In order to introduce the mathematical statement of the PBE problem let us start by defining the geometry of the domain where the problem is set. To that end, consider the schematic 2D representation of Figure 1 and let the computational domain Ω be defined as $\Omega \equiv \text{Int}\left(\overline{\Omega}_m \bigcup \overline{\Omega}_s\right), \Omega \subset \mathbb{R}^d$, where the interior of the molecule is denoted by Ω_m, and the solvent by Ω_s.
While solving the equation, both the solute and the solvent are treated as continuum media, each one characterized by its dielectric coefficient.
For a proper description of the problem physics, at least two different values for the coefficient are necessary. The difference between the two is due to the fact that the solvent is more easily polarized by an electric field, so that it is generally characterized by a higher dielectric constant ε_s, while the solute dielectric constant ε_m is about two orders of magnitude smaller.
Usually the relative electric permittivity is assumed to take the form

$$\begin{cases} \varepsilon_r(x) = \varepsilon_s(x) & \text{in } \Omega_s \\ \varepsilon_r(x) = \varepsilon_m & \text{in } \Omega_m \end{cases} \tag{1}$$

According to (1), the relative electric permittivity is uniform inside the solute, whereas it is a smooth function of the position x in Ω_s. This implies that ε_r is discontinuous across the interior boundary $\Gamma = \partial\Omega_s \cap \partial\Omega_m$ which represents the *molecular surface*.

The electrostatic potential is related to the *fixed charge density* ρ^f and to the *density of solution ions* ρ^s by the Poisson equation

$$\begin{cases} -\nabla \cdot (\varepsilon_0\ \varepsilon(x)\ \nabla\varphi(x)) = \rho^s + \rho^f & \text{in } \Omega \\ \varphi(x) = g(x) & \text{on } \partial\Omega, \end{cases} \tag{2}$$

where φ is the unknown scalar electrostatic potential and ε_0 is the electric permittivity of vacuum.
The PBE is obtained by assuming that the density of charge in the solvent obey an equilibrium Boltzmann distribution, so that it can be expressed as a function of the electrostatic potential.
In the most simple case (where only monovalent ions are assumed) such function may be expressed as

$$\rho^s = -2eC^0 \sinh\left(\frac{e\ \varphi}{k_B T}\right), \tag{3}$$

where k_B is the Boltzmann constant, T is the absolute temperature and e is the electron charge. The bulk concentration of monovalent species C^0, is often expressed in terms of the *ionic strength* I as

$$C^0 = 1000 N_{AV} I,$$

where N_{AV} is the Avogadro number and I is expressed as a molar concentration.
Substituting (3) into (2) and assuming the fixed charge density to be given by

$$\rho^f = \sum_i q_i\ \delta(x - x_i), \tag{4}$$

where the points x_i are the centers of the atoms composing the molecule, and the values q_i their charges, we end up wit the semilinear PBE

$$\begin{cases} -\nabla \cdot (\varepsilon_0\ \varepsilon_r(x)\ \nabla\varphi(x)) + 2eC^0 \sinh\left(\frac{e\ \varphi}{k_B T}\right) = \sum_i q_i\ \delta(x - x_i) & \text{in } \Omega \\ \varphi(x) = g(x) & \text{on } \partial\Omega. \end{cases} \tag{5}$$

The boundary condition $g(x)$ in (5) can be set in different ways and we refer the interested reader to, *e.g.*, [27], in what follows we will mostly assume, for sake of simplicity, that $g(x) = 0$.
If $\varphi(x)$ is not too large w.r.t. $k_B T$, the semilinear PBE can be linearized by approximating the sinh term with a linear function, which leads to the following *linearized* PBE

$$\begin{cases} -\nabla \cdot (\varepsilon_0\ \varepsilon_r(x)\ \nabla\varphi(x)) + 2\left(\frac{e^2}{k_B T}\right) C^0 \varphi = \sum_i q_i\ \delta(x - x_i) & \text{in } \Omega \\ \varphi(x) = g(x) & \text{on } \partial\Omega. \end{cases} \tag{6}$$

The major problems to face while trying to solve (6) are the following:

1. the very large scale algebraic systems of equations, due to the complex structure of the examined molecules, which require the adoption of HPC techniques for the solution of the problem;
2. the discontinuity of the dielectric coefficient across the interface between the solvent and molecule;
3. the presence of singularities, which can generate singularities in the problem solution and coefficients.

In order to deal with all these aspects, we adopted some mathematical and numerical tools and approaches that will be described in detail in the following sections.

2.1 Scaling and Dimensionless Formulation

Before proceeding to solve either (5) or (6) it is useful common practice to perform a scaling procedure to transform them into their nondimensional counterpart. The purpose of such procedure is, on one hand, to scale the variables and coefficients in the problem so that their numerical values are more amenable to numerical representation while, on the other hand simplifying the notation by grouping several dimensional coefficients into few dimensionless numbers.
To perform the scaling, we start by expressing each physical quantity in (2) as the product of a *scaling factor* (with physical dimensions, identified in what follows by the $\bar{\cdot}$ symbol) and a *dimensionless* variable denoted by the $\hat{\cdot}$ symbol, *i.e.*:

$$\begin{aligned} \varphi &:= \bar{\varphi}\hat{\varphi} \\ x &:= \bar{x}\hat{x} \\ \rho^s &:= \bar{\rho}\hat{\rho}^s \\ \rho^f &:= \bar{\rho}\hat{\rho}^f. \end{aligned} \tag{7}$$

Scaling Factor	Value
$\bar{\varphi}$	$k_B T/e$
$\bar{x}$	1 Å
$\bar{\rho}$	$e/(4\pi\bar{x}^3)$

Table 1: Choice of scaling factors

Table 1 shows the most common choices for the scaling factors, which are used in the implementation of `easy_pbe`.

Using the notation in (7), dividing both sides of (2) by $\bar{\rho}$, and assuming $\bar{\varphi} = \dfrac{k_B T}{e}$ (which is the most common choice for the scaling factor for electrostatic potential) leads to

$$-\frac{\varepsilon_0\ \bar{\varphi}}{\bar{x}^2 \bar{\rho}}\ \hat{\nabla}\cdot\left(\varepsilon_r\ \hat{\nabla}\hat{\varphi}\right) - \hat{\rho}^s = \hat{\rho}^f, \tag{8}$$

which, upon defining

$$\hat{\varepsilon}_0 := \frac{1}{\bar{\rho}\ \bar{x}^2}\ \varepsilon_0\ \bar{\varphi}$$

becomes

$$-\hat{\nabla}\cdot\left(\hat{\varepsilon}_0\ \varepsilon_r\ \hat{\nabla}\hat{\varphi}\right) - \hat{\rho}^s = \hat{\rho}^f. \tag{9}$$

Introducing the *scaled ionic force coefficient* $\hat{I} := \frac{2eC^0}{\bar{\rho}}$, letting $\hat{\varepsilon}_0\ \varepsilon_r \kappa^2 := \hat{I}$ and the nondimensional atomic charges $\hat{q}_i = \frac{q_i}{\bar{x}^3 \bar{\rho}}$ the semilinear version of the PBE can now be written as

$$-\hat{\nabla}\cdot\left(\hat{\varepsilon}_0\ \varepsilon_r\ \hat{\nabla}\hat{\varphi}\right) + \hat{\varepsilon}_0\ \varepsilon_r \kappa^2 \sinh\left(\hat{\varphi}\right) = \sum_i \hat{q}_i\ \delta(\hat{x} - \hat{x}_i)$$

while the linearized, nondimensional form becomes

$$-\hat{\nabla}\cdot\left(\hat{\varepsilon}_0\ \varepsilon_r\ \hat{\nabla}\hat{\varphi}\right) + \hat{\varepsilon}_0\ \varepsilon_r \kappa^2 \hat{\varphi} = \sum_i \hat{q}_i\ \delta(\hat{x} - \hat{x}_i). \tag{10}$$

For the sake of easing the notation, we will henceforth drop the $\hat{}$ symbol and consider the PBE to be expressed in nondimensional form.

2.2 Treatment of Point Sources

As anticipated in the previous section, the presence of point sources induces singularities in the analytical solution of the PBE that are one of the main issues in the numerical solution of the problem.

Two main approaches can be used to deal whith suche issue, namely (*i*) adopting a *regularized* reformulation of the problem (10) that removes the point sources by accounting for their presence by means of a surface charge density either on Γ or on $\partial\Omega$; or (*ii*) adopting a *smoothed* representaion of the atomic charges, *i.e.* representing them as finite charge density distribution functions with a small and compact, yet finite, support.

The first approach can be pursued in different ways that all exploit the availability of explicit expressions for the Green functions for the differential operator in (10), a review of a variety of such ways is given in [21], here we only give and example of such approach in Section 2.2.2, which is not currently implemented in `easypbe`.

The approach (*ii*) can also be implemented in different ways, but the most natural one in the framework of `easy_pbe` relies on expressing the atomic charges as linear combinations of Finite Element shape functions and is described in Section 2.2.1 below.

2.2.1 Representing Atomic Charges via FEM Shape Functions

As `easy_pbe` adopts a Finite Element discretization for tsolving the PBE, the most natural way of defining a continuous function with finite support consists of representing it as a linear combination of Finite Element shape functions.

In particular, as `easy_pbe` use continuous first-order Lagrangian Finite Elements, the degrees of freedom of each shape function are collocated at one of the cell vertices, therefore the natural choice consists of defining the i–th smoothed atomic charge density function $\rho_i(x)$ as

$$\rho_i(x) := \sum_k N_k(x) q_{ik},$$

where $N_k(x)$ denotes the shape function relative to the k–th mesh vertex and q_{ik} are the values of ρ_i at that vertex.
The main constraint that the nodal values q_{ik} must obey is that of preserving the total amount of charge, *i.e.*

$$\int_\Omega \rho_i(x)\ \mathrm{d}\omega = \sum_k q_{ik} \int_\Omega N_k(x)\ \mathrm{d}\omega = q_i.$$

A simple choice that satisfies this latter constraint is setting

$$q_{ik} := q_i \frac{N_k(x_i)}{\int_\Omega N_k(x)\ \mathrm{d}\omega}.$$

Indeed, from this definition it follows that

$$\int_\Omega \rho_i(x)\ \mathrm{d}\omega = q_i \sum_k N_k(x_i) \frac{\int_\Omega N_k(x)\ \mathrm{d}\omega}{\int_\Omega N_k(x)\ \mathrm{d}\omega} = q_i \sum_k N_k(x_i) = q_i,$$

where the last inequality discends from the fact that the basis functions enjoy a *partition of unity* property.
Notice that, for first-order continuous Finite Elements, if the atom center x_i lies in a cell τ, only the basis functions that are collocated at the vertices of τ are non vanishing at x_i therefore the weights q_{ik} are non vanishing only for a small set of values of k[1].

2.2.2 Example of Regularized Formulation Without Point Charges

The point charges can be eliminated from the problem introducing one of the so-called *regularization techniques*. We consider one example of these techniques which transfers the effect of the point charges to the boundary conditions and to the interface condition on Γ. Let's start considering the nondimensional semi-linear boundary value problem with homogeneous Dirichlet boundary conditions on $\partial\Omega$, namely

$$\begin{cases} -\nabla \cdot (\varepsilon_0 \varepsilon_r(x)\ \nabla\varphi(x)) = \ -\varepsilon_0 \varepsilon_r(x) \kappa^2 \varphi(x) + \sum_i q_i\ \delta(x - x_i) & \text{in } \Omega \\ \\ \varphi = 0 & \text{on } \partial\Omega \end{cases} \tag{11}$$

We decompose $\varphi(x)$ as $\varphi(x) = \varphi_s(x) + \varphi_m(x)$, where $\varphi_m(x)$ is the contribution to the electrostatic potential due only to point sources, while $\varphi_s(x)$ accounts for the correction due to the surface polarization charge related to the discontinuity in the permittivity. If the dielectric permittivity were independent from position x, the scalar electrostatic potential due to the i-th point charge and $\varphi_m(x)$ would be respectively

$$\varphi_i(x) = \frac{q_i}{4\pi\varepsilon_0\varepsilon_m\ |x - x_i|}$$
$$\varphi_m(x) = \sum_i \varphi_i(x)$$

Then the potential correction $\varphi_s(x)$ is given by the difference between the exact electrostatic potential and the one only due to point source charges

$$\varphi_s(x) = \varphi(x) - \varphi_m(x). \tag{12}$$

Since no point sources are present in Ω_s, the equation in the solvent domain becomes

$$-\nabla \cdot (\varepsilon_0 \varepsilon_m\ \nabla\varphi_m)(x) = -\varepsilon_0\varepsilon_m\ \Delta\varphi_m = 0, \text{ in } \Omega_s \tag{13}$$

We start from the equation in (11) restricted to the solvent domain Ω_s. Exploiting both the observation made in (13) and the definition of the potential correction (12), we obtain

$$\begin{aligned} -\nabla \cdot (\varepsilon_0 \varepsilon_s(x) \nabla\varphi_s) &= -\nabla \cdot (\varepsilon_0 \varepsilon_s(x) \nabla(\varphi - \varphi_m)) \\ &= -\nabla \cdot (\varepsilon_0 \varepsilon_s(x) \nabla\varphi) + \varepsilon_0 \varepsilon_s \Delta\varphi_m + (\varepsilon_0 \nabla\varepsilon_s) \cdot (\nabla\varphi_m) \\ &= -\varepsilon_0 \varepsilon_s \kappa^2 (\varphi_s + \varphi_m) + (\varepsilon_0 \nabla\varepsilon_s) \cdot (\nabla\varphi_m). \end{aligned}$$

[1]For a tensor-product hexahedral cartesian mesh, this would be exactly 8 values of k, but given that we use a refined mesh with hanging nodes the number of non vanishing coefficients depend on the mesh refinement.

While at the interface Γ

$$0 = [[-\varepsilon_r \nabla\varphi \cdot n]]_\Gamma = [[-\varepsilon_r \nabla\varphi_s \cdot n]]_\Gamma - [[\varepsilon_r]]_\Gamma (\nabla\varphi_m)|_\Gamma \cdot n,$$

where $[[\cdot]]_\Gamma$ represents the jump of quantity between brackets across the interface.

Finally, on the domain boundary $\partial\Omega$, substituting the definition of the potential correction (12), inside the Dirichlet boundary conditions of problem (11) we obtain

$$\varphi(x)|_{\partial\Omega} = (\varphi_s(x) + \varphi_m(x))|_{\partial\Omega} = 0.$$

Therefore, reformulating the problem (11) w.r.t. $\varphi_s(x)$, it becomes

$$\begin{cases} -\nabla \cdot (\varepsilon_0 \varepsilon_s(x)\ \nabla\varphi_s) = (\varepsilon_0 \nabla\varepsilon_s) \cdot \sum_i \nabla\Big(\frac{q_i}{4\pi\varepsilon_0\varepsilon_m|x-x_i|}\Big) - \varepsilon_0\varepsilon_s\kappa^2\Big(\varphi_s + \sum_i \frac{q_i}{4\pi\varepsilon_0\varepsilon_m|x-x_i|}\Big) & \text{in } \Omega_s \\ -\Delta\varphi_s = 0 & \text{in } \Omega_m \\ [[-\varepsilon_r \nabla\varphi_s \cdot n]]_\Gamma = [[\varepsilon_r]]_\Gamma \nabla\varphi_m \cdot n & \text{on } \Gamma \\ \varphi_s(x) = \sum_i \frac{q_i}{4\pi\varepsilon_0\varepsilon_m|x-x_i|} & \text{on } \partial\Omega \end{cases}$$

which has no point charges. Notice that third relation of the above system is expressed in $[C/m^2]$ SI units, so that it can be interpreted as a surface charge distribution. The main advantage of this regularization technique is to avoid the need of using small mesh sizes in Ω_m, but only at the molecular surface.

3 Geometric Representation of the Molecular Surface

The representation of the molecular surface Γ is a focal point of research in numerical methods for implicit solvent continuum models of biomolecular systems.
Given the positions x_i, $i = 1 \dots n_a$ of the atomic centers and the atomic radii r_i, one has to define the 2D manifold Γ, and this can of course be accomplished in many different ways, the choice among which is guided by several characteristics of the generated surface.
The main requirement is, of course, the accuracy of the electrostatic computation results, which is usually measured by comparison to results of solvent explicit computations. In addition to this, though, other properties are to be taken into account such as, for example, the differentiability of the surface w.r.t. to the atomic positions and radii and the continuity of the molecular surface area w.r.t. the atom positions.
The arguably most simple approach to the surface definition is the so called *Van der Waals* approach, according to which the molecular domain is defined as the union of the spherical regions centered at x_i and with radius r_i.
In [13], a comparison of three approaches is given, namely

1. *Blobby surface*,
2. *Skin surface*,
3. *Solvent Excluded Surface* (SES), a.k.a. Connolly surface.

These three options are all implemented in `easy_pbe`, the first one via a native implementation, the latter two via an interface to the geometrical library `NanoShaper` [14] and are therefore described in detail below.
library [12] to deal with this task.

3.1 Blobby Surface

Blobby surface, proposed in [7, 36], is an implicit surface whose definition is based on Gaussian atom center idea. Any implicit surface can be visualized using a ray-tracing technique which consists in finding all the points where an intersection with the surface occurs by tracing rays along a prescribed direction.
Its mathematical definition is quite simple and intuitive: it is the level set S, see Figure 2, of a particular function $G(x)$, namely

$$S := \{x \in \mathbb{R}^3 : G(x) = 1\}, \qquad G(x) = \sum_{i=1}^{n_a} \exp\Big(B\Big(\frac{||x-x_i||^2}{r_i^2} - 1\Big)\Big), \tag{15}$$

where x represents a point over the surface, while B is the *blobbyness*, a parameter that controls the surface roughness and that has to be tuned. Then there are three quantities that describe the chosen molecule: n_a, the

number of atoms inside the molecule, r_i which is the radius of the i-th atom and x_i, the vector of coordinates of the i-th atom center.

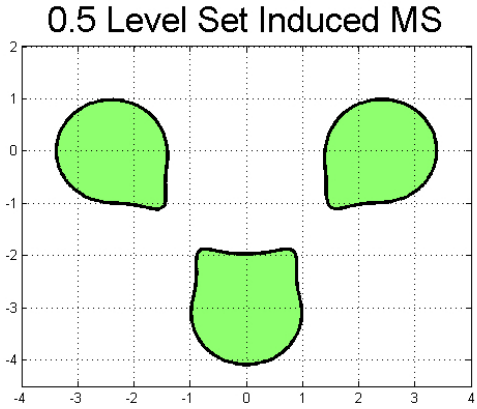


Figure 2: 0.5 level-set, Blobby surface [12].

Even if the usage of this kind of surface is fascinating because, due to its simple definition, it is easy to implement, also in parallel, it is not so easy to define a projection method for a point over the surface. Indeed, contrary to the Skin and SES surfaces, it cannot be partitioned in analytical patches. Another thing to point out is that while on one side it is appreciated for its smoothness which guaranties continuity and differentiability without any self-intersection nor singularities, on the other side it is not superior to the Connolly surface for the solution of electrostatic problems. Moreover, the choice of the *blobbyness*, which is a key parameter for the calculus of the energy estimates of the molecules, can create some problems. All these drawbacks lead to prefer Skin and SES surface for more accurate results.

3.2 Skin Surface

The Skin surfaces, as the Blobby, is an implicit surface appreciated for its smoothness and because it is expected to provide a surface area continuous with respect to atom positions and radii. Its definition was proposed in [15], and it is more complex from a mathematical point of view. It can be constructed starting from a set of weighted points S and a shrink factor $s \in (0, 1]$. The latter is the characterizing parameter for the Skin surface that has to be set, similarly to the *blobbyness* B in the Blobby case. The set S and the weights of the points w_i can be defined as follows:

$$\begin{gathered} S = \left\{ p_i = (c_i, w_i),\ x_i \in \mathbb{R}^3,\ \ i = 1, ..., n_a \right\} \\ w_i = \frac{r_i^2}{s} \end{gathered} \tag{16}$$

where n_a is the number of atoms inside the considered molecule and r_i and x_i are respectively the radius and the center of the i-th atom.
For defining precisely the Skin surface, the concept of *mixed complex* has to be introduced, since starting from it the surface equations can be derived. The *mixed complex* is composed of *mixed cells* μ_X^s which are solids that can be computed through the Minkowsky sum "$\oplus$", as reported in (17), exploiting either the Voronoi diagram or Delaunay tetrahedrization.

$$\mu_X^s = \{ s \cdot \nu_x \oplus (1 - s) \cdot \delta_x \} \tag{17}$$

In this definition, X is a subset of a finite set of spheres in $\mathbb{R}^3$ and ν_X is a non empty Voronoi cell with corresponding Delaunay simplex δ_X [15]. After the *mixed cells* are calculated, a ray-tracing software can be used in order to define the inside and outside of the surface. A schematic representation of both the *mixed complex* and Skin surface of a simple molecule where 8 equal atoms are put on the vertices of a cube can be seen in Figure 3.

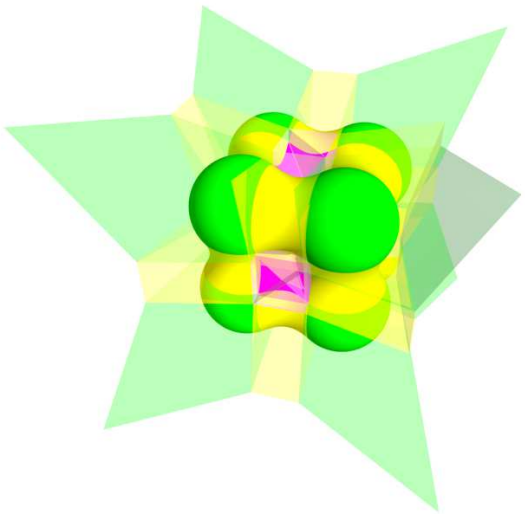
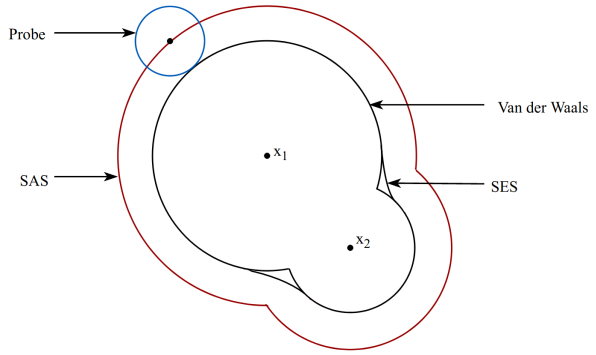


Figure 3: Skin Surface and mixed complex (transparent). Surfaces corresponding to $\mu^s_{3,0}$, $\mu^s_{2,1}$, $\mu^s_{1,2}$ *mixed complex* are represented in green, yellow and magenta, respectively [12].

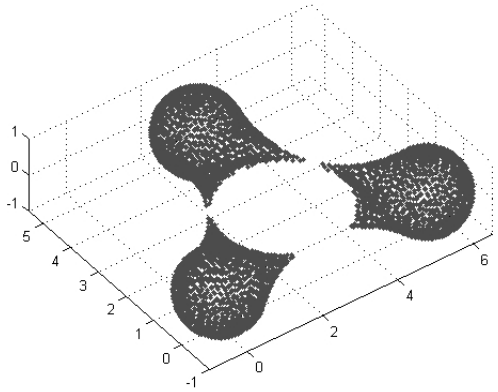
The existence of such fast algorithms as Voronoi diagram or Delaunay tetrahedrization for the construction of the Skin surface constitutes an advantage with respect to other kind of surfaces. In addition, also the presence of pathological configurations which can create normal discontinuities are extremely limited. In any case, the computational costs are higher than the Connolly ones and, moreover, Skin surface can create unphysical high dielectric cavities, regions that can affect the energy estimate of the system.

3.3 Connolly Surface

The SES is one of the most commonly adopted molecular surfaces in Computational Biological Chemistry for the visualization of the surface area. This kind of molecular surface was suggested by Lee and Richards in [22] but it is commonly known as Connolly surface, so called because of the famous algorithm computation proposed by Connolly.
For defining the Connolly surface, we start from a simpler definition of molecular surface, in which it is defined as the contact surface between the Van der Waals envelope of the solute molecule and the probe solvent molecule. The Van der Waals surface is the enclosing surface obtained by the superposition of hard spheres which represent the atoms present inside the examined solute molecule. The union of all the hard spheres forms the Van der Waals volume and each of those spheres has radius equal to the Van der Waals radius, that is calculated as a distance between atoms in crystals and indicates the largest distance at which an atom repels its neighbors. On the other hand, considering the solvent molecules, Solvent Accessible Surfaces (SAS) are adopted. This kind of surface can be described as the locus of the centers of a spherical probe that rolls over the molecular system. This kind of surface coincides geometrically with the Van der Waals surface, where each radius of the above described hard spheres is increased by the probe radius size. Starting from SAS, further developments lead to the Solvent Excluded Surface (SES), which separates the finite volume accessible to a finite size solvent probe from the inaccessible one. Indeed, also SES surface is computed by rolling the probe sphere over the van der Waals surface of the molecule, but in this case the surface is determined by the surface of the probe, instead of its center. Figure 4a shows schematically the difference between SAS, SES anche the Van der Waals surface, while Figure 4b represents a simple example of Connolly surface in 3D.



(a) Schematic representation of the difference between SES, SAS and Van der Waals surface, with an example of probe.



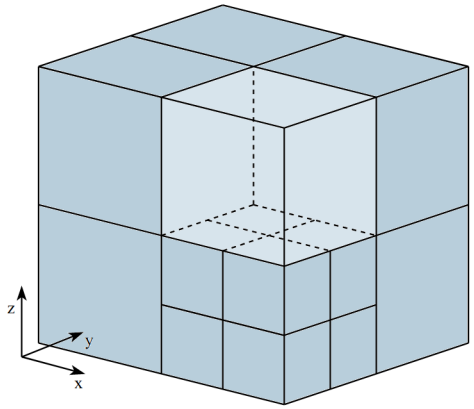
(b) Connolly surface [12].

Figure 4

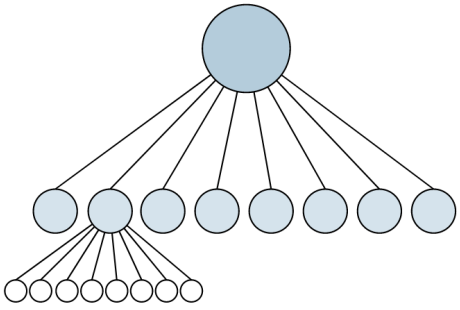
The Connolly surface is appreciated because it is conceptually simple, but on the other hand it can present some singularities and a discontinuous dependence of the surface area on the position of atom centers.

4 Oct-tree grids

The very large size and complex structure of the molecules being studied leads to very large scale algebraic systems of equations which demand for state-of-the-art HPC solver technology. The novelty of this work w.r.t. the already existing PBE solvers consists in solving the linearized Poisson–Boltzmann equation with Finite Element method on Oct-tree, which are hierarchically refined, non conforming, cartesian grids. Indeed, typically the other solvers calculate the electrostatic potential adopting either tensor product cartesian grids with simple numerical methods, which is characterized by high memory efficiency, but it is not so accurate, or adaptive, simplicial, conforming meshes with Finite Differences method, less efficient and more accurate. The choice of Oct-tree grids gives our solver benefits in terms of efficient parallel partitioning, refinement, coarsening and balancing, while FE is more flexible for local mesh refinement.
The structure of an Oct-tree grid can be easily represented as a tree (Figure 5b), as the name suggests, where each octant (*i.e.* each element of the mesh, see Figure 5a) is a either a leaf or has 8 children. Indeed, the mesh is generated starting from a 3D cube and then iteratively refining or coarsening the elements in a non conforming way.



(a) Example of a simple Oct-tree mesh.



(b) Tree associated to the Oct-tree grid.

Figure 5

More in detail, the starting 3D cubic domain is initially uniformely refined in a rough manner in order to obtain a first solution, that will be used as a starting point for the following refine-coarsen iterations. Then, the idea is to divide a mesh element into 8 smaller cubes when it contains either an atom charge or part of the molecular surface, and coarsen the grid otherwise.

The initial level of refinement is indicated as level 0, then if one element belonging to level 0 is refined N times, each of the resulting subelements belongs to level N. Each cubic element of the mesh is characterized by the same numbering of faces, edges and nodes, which is the one reported in Figure 6. All these enumerations, jointly with the quadrant indexing too, follow the z-ordering.

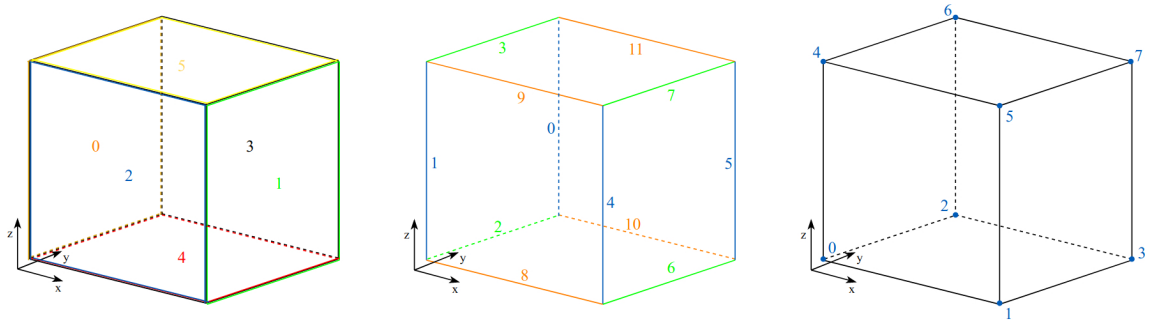
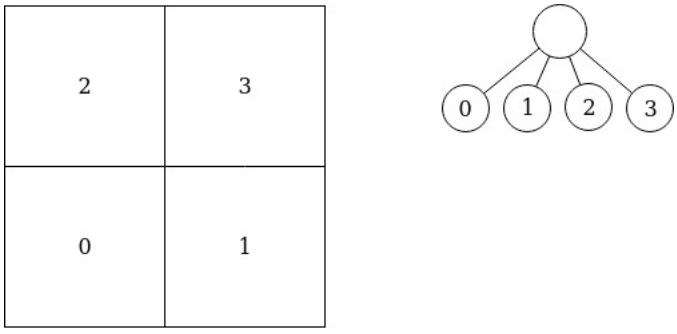
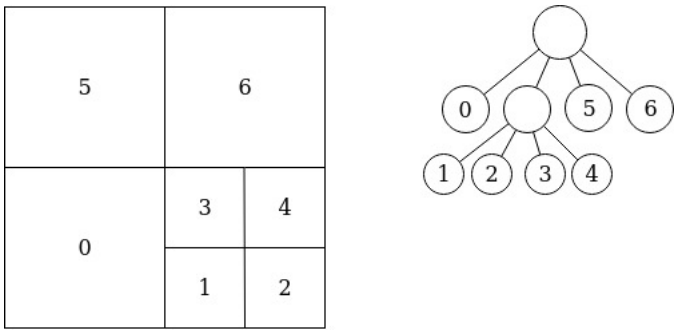


Figure 6: From left to rigth: numbering of faces, edges, nodes of each quadrant.

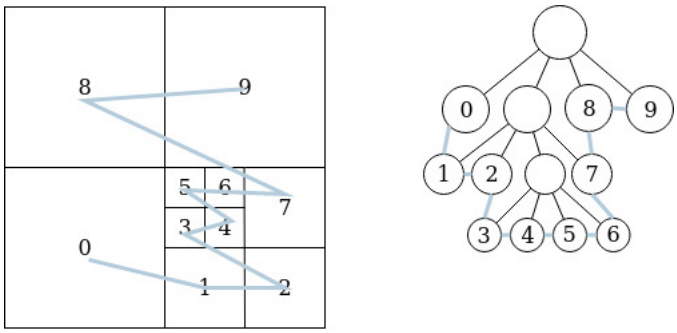
The quadrant numbering is particularly important in parallel applications because it produces a smart subdivision of the mesh octants among the involved processors. This operation takes the name of parallel partitioning and is performed according to a given number of elements per processor or according to some prescribed weights associated to those elements. To better understand in which way this is done, a simple example of the refinement process of a 2D Quad-tree mesh is reported below in Figure 7. The initial square domain is split into four elements, all belonging to level 1 (Figure 7a). Then, only the element indicated with number 1 is refined and all the squares are renumbered as shown in Figure 7b. In the third step element 3 is divided into four subelements so that the enumeration changes again. The correlation between the quadrant numbering and the corresponding tree is pointed out in Figure 7c, where the Morton S-shaped space filling curve is reported. The nodes are subdivided among the processors following this curve, so that nodes belonging to the same processors are also neighbors in the mesh. This is useful in order to limit communications between processors.



(a) First refinement step.



(b) Second refinement step.



(c) Third refinement step with Morton S-shaped space filling curve.

Figure 7: Quad-tree mesh refinement example.

The resulting mesh is a non conforming mesh, in which at most 2:1 edge size relations between neighboring octant is ensured, that is the so-called balancing. This is done in order to properly deal with the hanging nodes. These kind of nodes are, in 2D case, nodes belonging to an edge which is refined for two elements, but not for the neighboring one, like for example two red marked nodes in Figure 8. In the Oct-tree cases these nodes can be generated both on faces and on edges. Their main characteristic is that they present a local numbering, but no global numbering. In order to be identified, they are related to the global numbering of their non-hanging parents, *i.e.* the nodes belonging to the same edge or face (the latter only in the 3D context) where the hanging node is present. Since hanging nodes are not present inside the set of the degrees of freedom, they have to be taken into account in a different way.

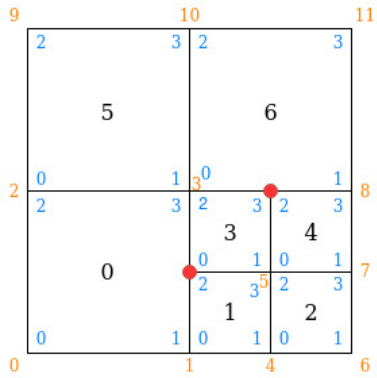


Figure 8: Example of hanging nodes, marked with red spots. Black numbers refer to quadrant numbering, blue to node local indices, orange to node global indices.

5 Implementation strategy

During this thesis we have developed a parallel solver on adaptive meshes for the solution of the linearized PBE. The implemented code can be found inside this repository: `https://github.com/carlodefalco/easy_pbe`. The user reference with the instructions on how to compile and use the code and for the output visualization can be found in Appendix A.

To achieve our goals, the `bim++` library which provides at the same time a high-level access to meshes with a series of functions for the solution to PDEs in the FE method context and also a `c++` interface to `p4est` library, was adopted. Through `p4est` library it was possible to handle the Oct-tree grids and perform the hierarchical mesh refinement over the octants that compose the 3D domain, while the efficient construction of the molecular surface and the evaluation of its interior and exterior points, rely on `NanoShaper`. The latter is able to find some surface characteristics such as the volume of the molecule, the surface area and cavities (regions present inside proteins, big enough to contain one or more water molecules) through a ray-casting technique. It is possible to construct both SES and Skin surfaces, while the Blobby surface is not available for the `NanoShaper API`. For this reason this kind of surface was directly implemented inside the code using its definition (15).

5.1 Code Description

The main function of the code is the `poisson_boltzmann.cpp` file, in which the functions and methods necessary for the computation of the electrostatic potential are called. The algorithm is composed of six steps, described in the subsections below.

5.1.1 Data Input

The first step consists in reading from the two input files selected by the user: the `options.pot` file, with the simulation parameters, and a .pqr file, containing the 3D molecular structure. The atom characteristic extracted from such file are stored in vector of `NS::Atom` class objects. The `Atom` class is defined in the `NanoShaper` library as reported below, where the member functions have been dropped for brevity.

```
class Atom: public Point
{
public:
        double radius;
        double radius2;
        double charge;
        int dielectric;
        AtomInfo ai;

        // *member functions*
};
```

5.1.2 Surface Creation

For the surface creation a query to `NanoShaper API` was introduced. This is done only by rank zero at the line

```
ray_cache.init_analytical_surf (pb.atoms, pb.surf_type, pb.surf_param,
                                pb.stern_layer, pb.num_threads);
```

where the analytical surface is created starting from the atom characteristics, passed to `NanoShaper` through `pb.atoms`, that is the previously cited `std::vector<NS::Atom>`.
Once the analytical surface is created, the geometric surface can be defined by tracing the rays which start from one domain boundary face and arrive to the other. Through this technique, `NanoShaper` is able to stabilize where the molecular surface intersections occurs. This operation is performed in the following step.

5.1.3 Mesh Creation and Refinement

The Oct-tree grid is created and initialized accordingly to the user requests, then the refinement and coarsening operations can be performed. In particular, at each iteration a loop over the octants of the mesh is done. The current cell is refined either if it contains at least one non hanging node inside the molecule and at least one in the solvent, which means the molecular surface is contained in the current element, or when it contains a point charge. During the coarsening iterations the opposite idea is carried out: the mesh can be coarse where neither the point charges nor molecular surface are present and the 2:1 edge size relation between neighboring octants is still respected.
In order to establish if a point is either inside or outside the molecule, we exploits the information about the intersection given by `NanoShapers` and assign to that point value 1 when it is inside the molecule, 0 if it is outside, -1 if unknown. The latter is returned only by the non-zero processors when the ray to which the point belongs is not present inside their `ray_cache_t` object. The design of the `ray_cache_t` class is explained in detail in the next paragraph.

5.1.4 Ray Caching

`NanoShaper` can be executed either in serial or using multiple threads, while our solver parallelization rely on MPI which is based, instead, on multi-core processing. This difference leads us to implement a Master–Slave program, where the `NanoShaper` functions are explicitly called only by rank zero, while the other processors ask for the desired information to the root. In order to limit the communication among the processors, we created a cache structure, consisting in a `std::map`, in which the rays progressively traced by `NanoShaper` and received from rank zero are collected. This can be done because the mesh is cartesian and hierarchically refined, which means that the number of traced rays can be at most equal to the number of rays casted for a uniform grid. The logarithmic complexity of the map find method can be even reduced using a `std::unordered_map` to linear complexity.
The rays map declaration is the following

```
std::map<std::array<double, 2>, crossings_t, map_compare> rays;
```

Each key of the map is an array of two double, storing the two coordinates corresponding to the traced ray, while each value is an object of the class `crossings_t`. The keys are sorted using the previously implemented `map_compare` functor. Since the rays are traced along the y direction, the key–array contains the x and z coordinates prescribing that ray. The `crossings_t` class is the one containing the ray information: the direction along which it is traced, the coordinates prescribing it, the vector containing the intersections with the molecular surface, the corresponding normals and the variable `init`, created to check if for this class object the intersection and normal fields have been already initialized.

The member functions of the `ray_cache_t` class involved in the caching operation are:

- **crossings_t & operator() (double x0, double x1)**;
- **void fill_cache ()**.

While the second of the two will be described in detail in the following section, here we focus on the first. It overloads the function call operator. We call such function by passing as arguments the two coordinates specifying the desired ray. There are two possible scenarios:

1. the map already contains the initialized ray and the correspondent `crossings_t` object is returned;
2. the map doesn't contain an initialized ray, in which case further distinctions have to be done. When the rank calling the operator is zero, the ray is created. initialized and returned. Instead, if rank is not zero, the requested ray is created, but cannot be initialized. The returned ray has `init` value equal to zero.

Below is reported part of the function call operator definition, in which the previously described distinction corresponds to the `if-else` conditional statements.

```
crossings_t &
ray_cache_t::operator() (double x0, double x1)
{

  //...
  static std::array<double, 2> start_point;
  start_point = {x0, x1};
  auto it0 = rays.find (start_point);

  if (it0 != rays.end () && it0->second.init == 1)
    cr_t = it0->second;

  else if (it0 != rays.end () && it0->second.init == 0)
    if (rank == 0)
      compute_ns_inters (it0->second);

  else
    {
      // create a new ray
      crossings_t tmp;
      tmp.point[0] = x0;
      tmp.point[1] = x1;
      if (rank == 0)
          compute_ns_inters (tmp);
      rays[start_point] = tmp;
      cr_t = rays[start_point];
    }

  return rays[start_point];
}
```

In order to check if the passed couple of values corresponds to a key of `rays`, the map `find` method is used. The function `compute_ns_inters` is a member function of the same class, responsible for the initialization of the rays through `NanoShaper`. In particular, it fills the `intersections` and `normals` vectors and change the variable `init` value from 0 to 1.
Regardless of the actual scenario, the function call operator returns a `crossings_t` object. When initialized, it is used to establish if a given point belonging to that ray is either inside of outside the molecular surface. Instead, if `init = 0`, the calling processor must ask to rank zero for the initialization of that ray and the couple given by x, z is added to the requested rays set, namely `rays_list`.

The `fill_cache` function is the second method involved in the caching process. Inside this function the non zero ranks ask and receive the missing rays from the root. If the requested ray is already present inside rank zero cache, then it is just transmitted to the asking processor, otherwise it is first calculated and then sent. Since all the involved objects belong to the standard library, the communication among the processors can be done using the *serialization* process. It consists in translating data structures or objects into a format that can be communicated in an efficient way. To transmit each data, a `write` function is implemented. It converts the object to a vector of bytes that can be sent as an MPI message. After this first communication, the corresponding `read` function transforms the vector of bytes into the initial object.
Since the `crossings_t` class is non–trivial, the serialization of the `rays` map was obtained taking advantage of the implementation of simpler serializations. We started from the serialization for vectors, maps and vector of maps containing trivial types (`serialization.h` file). Thereafter, we exploited them in order to implement the write and read methods for the `crossings_t` objects first, and `rays` map then.
In the following box, the principal lines of `fill_cache` function, where both the serialization and the communication are performed, are reported.

```
void
ray_cache_t::fill_cache ()
{
  //...
  std::vector<unsigned char> local_ser_rays_vec = serialize::write (rays_vector);
  //...
```

```
  MPI_Gatherv (&local_ser_rays_vec[0], ser_rays_len, MPI_CHAR, &global_ser_rays_vec[0],
              proc_ser_rays_len, displ_ser_rays_vec, MPI_CHAR, 0, mpicomm);

  if (rank == 0)
  {
    serialize::read (global_ser_rays_vec, rays_vector);

    //...
    for (int i = 1; i < size; i++)
      {
        //...
        std::transform (rays_vector.begin () + cum_rays_vec[i-1],
                        rays_vector.begin () + cum_rays_vec[i],
                        std::inserter(local_req_rays_map, end(local_req_rays_map)),
                        [this](std::array<double,2> arr)
                        { return (std::pair<std::array<double,2>, crossings_t>
                        (arr, (*this)(arr[0], arr[1]))); });

        local_ser_map = write_map (local_req_rays_map);
        //...
        global_ser_map.insert (global_ser_map.end (), local_ser_map.begin (),
                               local_ser_map.end ());
        //...
      }
    //...
  }

  //...
  MPI_Scatterv (&global_ser_map[0], proc_ser_map_len, displ_ser_map, MPI_CHAR,
                &local_ser_map[0], local_ser_map_len, MPI_CHAR, 0, mpicomm);

  if (rank != 0)
    {
      //...
      read_map (local_ser_map, local_req_rays_map);
      (this->rays).insert (local_req_rays_map.begin (), local_req_rays_map.end ());
    }
  //...
}
```

First of all, each list of the requested rays is converted to a vector of bytes using the `serialize::write` for vectors. Then, the `local_ser_rays_vec` are sent using `MPI_Gatherv` to rank zero, which collects them inside a global vector.
At this point, the root can transform back the received data into a `std::vector<std::array<double, 2»`. They will be used to fill an initially void map, previously created for each processor, with the requested rays just computed or found in the cache from rank zero. All this process is done within the `std::transform`, with the aid of a lambda function. Then, the serialization is performed by converting, sending and transforming back the local maps. Finally, the initialized rays are added to each processor cache and can be used to correct the previously unknown nodes marked with value -1.

5.1.5 Matrix and Vectors Assembling

Once the previous steps are completed, the matrix and vectors composing the linear system for the solution of the non-dimensional PBE can be assembled. This operation is performed by `bim++`, using the `bim3a_advection_diffusion`, `bim3a_reaction` and `bim3a_rhs` functions. The matrix and vectors are respectively a distributed matrix exported in `aij` or `csr` format, depending on which linear solver was selected, and distributed vectors.

5.1.6 Linear System Solution

Finally the electrostatic potential is calculated by solving the linear system using either `mumps` or `lis`. The first implements a multi-frontal direct method, which solves the system through the LU factorization, while the latter is a large library offering different iterative methods, in particular Krylov methods, with a large variety of preconditioners.

6 Validation Test Cases with Exact Solution

Before passing to real molecule simulations, we performed a series of validation tests with the aim of checking the correctness of results given by the implemented code. For these *ad hoc* simulations the exact solution can be easily computed and then compared with the numerical one using the Paraview software. We constructed a .pqr file (Figure 9) containing one charged atom with radius equal to 10Å and charge equal to the proton charge, posed at the origin of Cartesian axes. Six more atoms with null charges and 0.01Å radii are disposed symmetrically around the central charged particle, within its radius. These atoms are present in order to guarantee the construction of the molecular surface using NanoShaper, which requests at least four atoms inside the .pqr file.

```
ATOM      1  X    X      1       0.000   0.000    0.000  1.000 10.0
ATOM      2  X    X      1       0.000   0.000    0.010  0.000 1.00
ATOM      3  X    X      1       0.000   0.000   -0.010  0.000 1.00
ATOM      4  X    X      1       0.000   0.010    0.000  0.000 1.00
ATOM      5  X    X      1       0.000  -0.010    0.000  0.000 1.00
ATOM      6  X    X      1       0.010   0.000    0.000  0.000 1.00
ATOM      7  X    X      1      -0.010   0.000    0.000  0.000 1.00
```

Figure 9: File UNITSPHERE_DISC.pqr content.

The domain is the cube $\Omega = [-32, 32]^3$, while the resulting molecular surface is a sphere of radius 10Å, as schematically shown in Figure 10, where the six particles with null charge have been neglected for simplicity, because they don't affect neither the exact and numerical solutions calculation nor the resulting molecular surface.

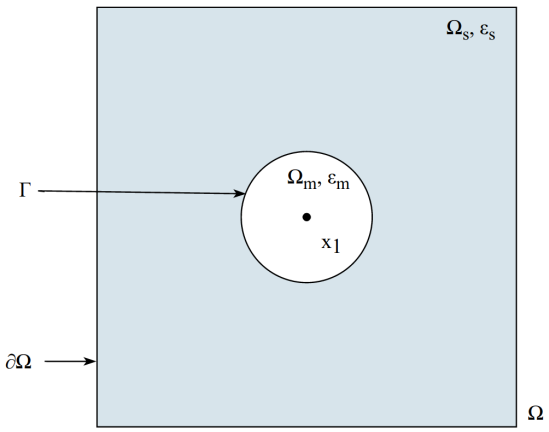


Figure 10: Schematic representation of the domain used for the validation tests.

6.1 Test Case in a Homogeneous Dielectric

The first test case was done to compare the exact solution with the one calculated with our code in a homogeneous dielectric. The problem geometry is the same reported in Figure 10, but the distinction between Ω_m and Ω_s refers only to the molecular surface which separates the solvent and the solute, not to the dielectric coefficients. Indeed, they are assumed to be both equal to the vacuum permittivity in all the domain Ω, which means $\varepsilon_m = \varepsilon_s = 1$. The unique charge is q located at the axis origin, which coincides with the atom center x_1.
For this simple test case, we calculated the exact electrostatic potential φ using Gauss's Law for the electric field. It states that the electric flux through any closed surface is proportional to the total electric charge enclosed by the surface, namely

$$\oint_S E \cdot \mathbf{n}\, ds = \frac{q}{\varepsilon_0 \varepsilon_m}, \tag{18}$$

where E indicates the electric field, S is the closed surface containing the charge q and $\mathbf{n}$ is the outward pointing unit normal at each point on the boundary.
In our case, the surface S is a sphere of radius r centered at the point charge q, which coincides with the proton charge. Exploiting the spherical symmetry, the electric field E becomes a constant that can be taken out of the integral. Then, integrating the remaining quantities we obtain Coulomb's law

$$E(r) = \frac{q}{4\pi\varepsilon_0\varepsilon_m r^2}. \tag{19}$$

Recalling the electrostatic potential definition

$$E = -\nabla\varphi$$

and substituting Coulomb's law (19), the equation we want to solve, in spherical coordinates, assumes the following expression

$$-\frac{\partial\varphi(r)}{\partial r} = \frac{q}{4\pi\varepsilon_0\varepsilon_m r^2}, \quad 0 < r < \infty. \tag{20}$$

Integrating both sides of the above relation we obtain the following electrostatic potential

$$\varphi(r) = \frac{q}{4\pi\varepsilon_0\varepsilon_m r} + C, \tag{21}$$

which is defined up to an additive constant. In order to find the value of the constant C, a position where the electrostatic potential is null have to be chosen. Assuming the potential null at infinity, namely

$$\lim_{r\to\infty} \varphi(r) = 0, \tag{22}$$

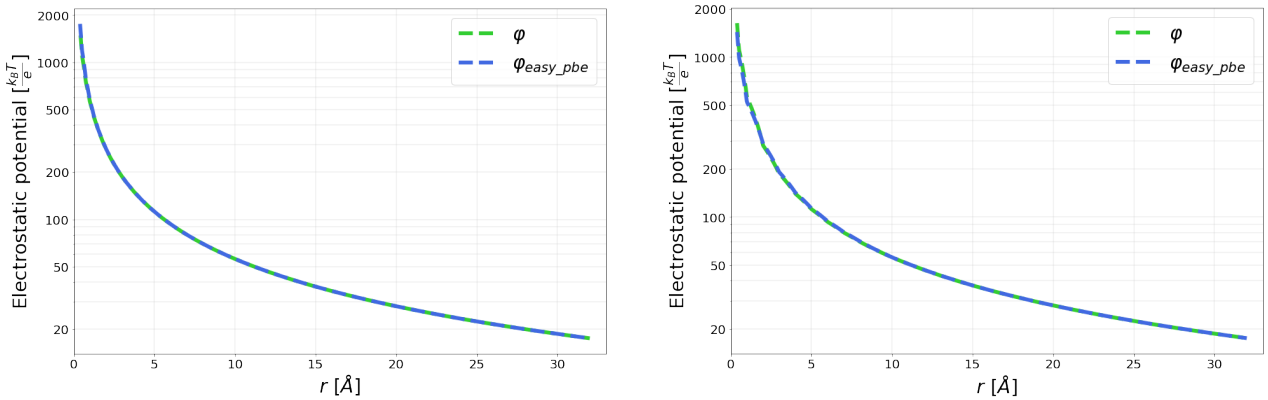
the constant C is equal to 0 and the exact solution is given by the Coulomb potential

$$\varphi(r) = \frac{q}{4\pi\varepsilon_0\varepsilon_m r}. \tag{23}$$

For what concerns the numerical solution with our code, we calculated the potential by fixing the ionic strength to zero, so that also the κ^2 coefficient is zero and the solved equation coincides with (20). The boundary conditions were manually set as follows

$$\varphi_{easy\_pbe}(r) = \varphi(r), \quad \text{on } \partial\Omega.$$

The comparison between the numerical and theoretical solutions is reported in Figure 11. We calculated the easy_pbe electrostatic potential using both a uniform grid and an adaptively refined grid. The former has with a discretization pass equal to 0.25Å, the latter has minimum pass of about 0.03Å and maximum pass 1Å. This results in a uniform grid which is made of about 16,8 millions of cells, while the refined one contains about 2,6 octants. Despite the large difference between the number of elements, in both cases the discrepancy from the exact solution is negligible.



(a) Uniformly refined Oct-tree grid of level 8.

(b) Adaptive Oct-tree grid with maximum level of refinement equal to 11, minimum level equal to 6.

Figure 11: Exact solution (green) vs numerical solution (blue) calculated with easy_pbe solver.

In Figure 12 the relative error of φ_{easy_pbe} calculated on the uniform grid nodes is represented. The curve rapidly drops to zero starting from the maximum value of about 17%. Indeed, the relative error becomes less than 1% from 1.5Å and less than 0.1% from 0.4Å. The numerical solution becomes nodally exact in some point starting from 21Å, in the solvent domain. The differences in the first part can be attributed to the imprecise geometry and to the source charge which is not precisely a point source for the solver.

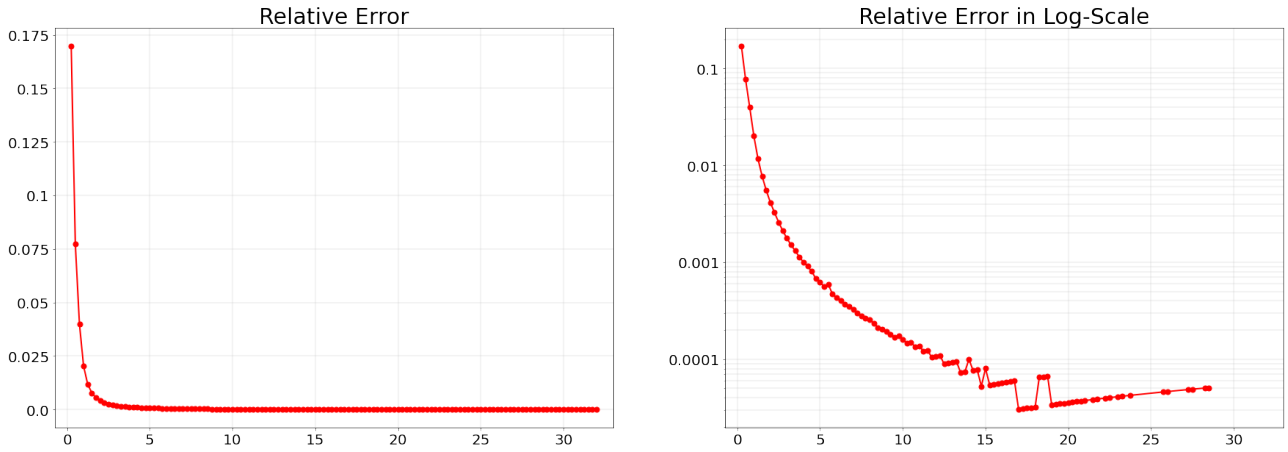


Figure 12: Relative error of the electrostatic potential obtained using the uniform grid.

6.2 Test Case in a Non Homogeneous Dielectric

In this second validation case we wanted to test if the electrostatic potential was correctly calculated when a dielectric discontinuity is present at the interface between the solvent and the solute. We compared the results obtained using the easy_pbe solver with both the exact theoretical solution and the solution given by the Delphi solver, where the computational domain Ω_{Delphi} was $[-25, 25]^3$ in both this and the following test cases. For easy_pbe instead, the problem setting is the same introduced in the previous sections and reported in Figure 10. Inside the sphere, we fixed the molecular relative permittivity coefficient $\varepsilon_m = 1$, while outside the molecule the solvent permittivity constant is equal to $\varepsilon_s = 40$. Like in the previous test case, the ionic strength was fixed to zero, so that no reaction term is present in the equation.
Focusing on the region outside the molecule, the system of equations takes the form reported below

$$\begin{cases} -\dfrac{1}{r^2}\dfrac{\partial}{\partial r}\left(\varepsilon_0\varepsilon_s r^2 \dfrac{\partial \varphi_{out}}{\partial r}\right) = 0, \quad r_s < r < \infty \\ -\left(\varepsilon_0\varepsilon_s \dfrac{\partial \varphi_{out}}{\partial r}\right)\Big|_{r_s} = \dfrac{q}{4\pi r_s^2} \\ \lim\limits_{r\to\infty} \varphi_{out}(r) = 0, \end{cases} \tag{24}$$

where r_s is the radius of the molecular surface sphere. The first equation in (24) is the Poisson's equation where the rhs is equal to zero because no charge is present in the solvent. The second and third equation of the system are the boundary conditions of the problem. The first of the two is prescribed at the interface between the solvent and the molecule, the second assumes the potential equal to zero at infinity. The analytical solution of this problem is given by

$$\varphi_{out}(r) = \frac{q}{4\pi\varepsilon_0\varepsilon_s r}. \tag{25}$$

Inside the molecular surface we can apply Gauss's Law for the electric field and use the electrostatic potential definition as before, which means that the equation to be solved is (20), with the corresponding solution (21). The constant C can be found imposing the continuity of the electrostatic potential at the interface

$$\varphi_{in}(r_s) = \varphi_{out}(r_s). \tag{26}$$

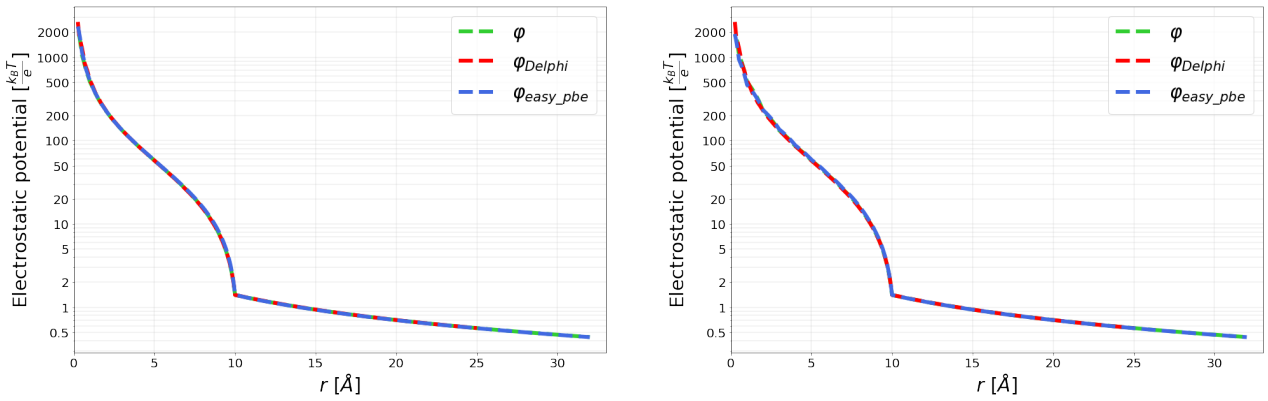
We obtain

$$\varphi_{in}(r) = \varphi_{out}(r_s) + \frac{q}{4\pi\varepsilon_0\varepsilon_m r}\frac{(r_s - r)}{r_s}. \tag{27}$$

The exact electrostatic potential in all the domain Ω is the sum of the two local solutions (27) and (25) restricted to their region of interest

$$\varphi(r) = \varphi_{in}(r)\chi_{\Omega_m}(r) + \varphi_{out}(r)\chi_{\Omega_s}(r). \tag{28}$$

Then, we compared our result with the analytical solution and the numerical solution given by the Delphi solver. We imposed manually the analytical solution value at the boundary of the easy_pbe potential, *e.g.* $\varphi_{easy\_pbe}(r)|_{\partial\Omega} = \varphi(r)|_{\partial\Omega}$. In Delphi we don't have the possibility to set manually the desired value at the boundary, so we choose *Coulombic* boundary conditions. This option gives an estimate of the potential at a certain distance. For the Poisson's equation, this value is equal to the potential generated by a point charge in a homogeneous dielectric with permittivity constant equal to ε_s, which is exactly $\varphi(r)|_{\partial\Omega}$. The results are reported in Figure 13, where φ_{easy_pbe} was calculated both on a uniform grid composed of about 16 millions of cells and a refined mesh with less than 6 millions of elements. Also in this case the graphs are very similar so that it is difficult to see the differences and the discontinuity at 10Å is always well detected.



(a) Uniformly refined Oct-tree grid of level 8.

(b) Adaptive Oct-tree grid with maximum level of refinement equal to 11, minimum level equal to 6.

Figure 13: Comparison between exact solution (green), easy_pbe solution (blue) and Delphi solution (red).

6.3 Test Case in a Non Homogeneous Dielectric with Reaction term

In this last validation test the problem setting is the same of the previous validation case, where the two relative dielectric coefficients are $\varepsilon_m = 1$ and $\varepsilon_s = 40$. The only difference lies in the ionic strength, which is here fixed equal to the commonly accepted value of 0.145M. In this case the Poisson's equation becomes the PBE, where also the reaction term is present. We checked the correctness of our code by comparing the easy_pbe potential with the analytical solution first, and then with Delphi's results.
Similar arguments to the ones made for the test without the reaction term can be done also in this situation. This means that, outside the molecule, the problem for the electrostatic potential becomes

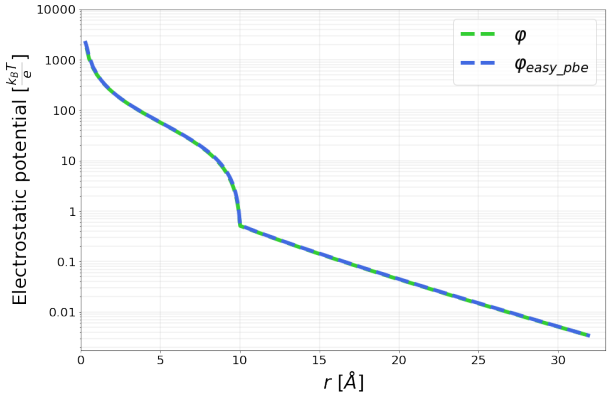
$$\begin{cases} -\dfrac{1}{r^2}\dfrac{\partial}{\partial r}\left(\varepsilon_0\varepsilon_s r^2 \dfrac{\partial\varphi_{out}}{\partial r}\right) + \varepsilon_0\varepsilon_s\,\kappa^2\varphi_{out} = 0, \quad r_s < r < \infty \\ -\left(\varepsilon_0\varepsilon_s \dfrac{\partial\varphi_{out}}{\partial r}\right)\Big|_{r_s} = \dfrac{q}{4\pi r_s^2} \\ \lim\limits_{r\to\infty}\varphi_{out}(r) = 0. \end{cases} \tag{29}$$

The solution translates into

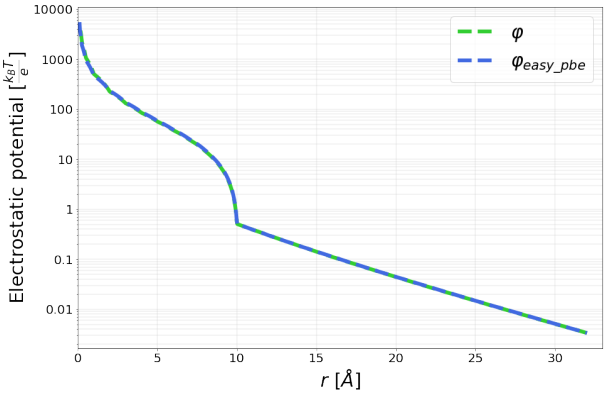
$$\varphi_{out}(r) = \frac{q}{4\pi\varepsilon_0\varepsilon_s r}\frac{e^{\kappa(r_s - r)}}{1+\kappa r_s}. \tag{30}$$

The problem in the solute is given by equation (20) coupled with (26). The solution is given by (27), with $\varphi_{out}(r_s)$ calculated from (30). The analytical solution is composed of the two contributions (28).

Numerically, we imposed $\varphi_{easy\_pbe}|_{\partial\Omega} = \varphi|_{\partial\Omega}$. The uniform and refined grids are characterized by the same number of cells reported for the previous test case and the results are close to the analytical solution as shown in Figure 14.



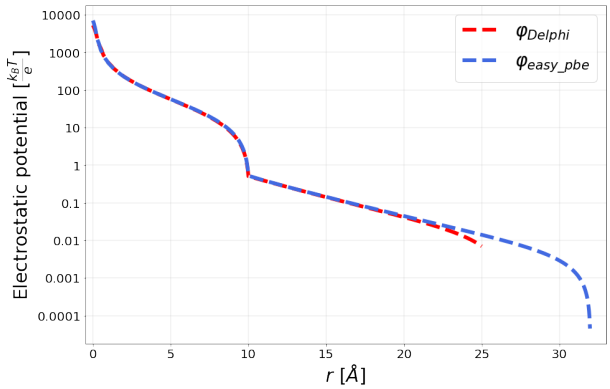
(a) Uniformly refined Oct-tree grid of level 8.



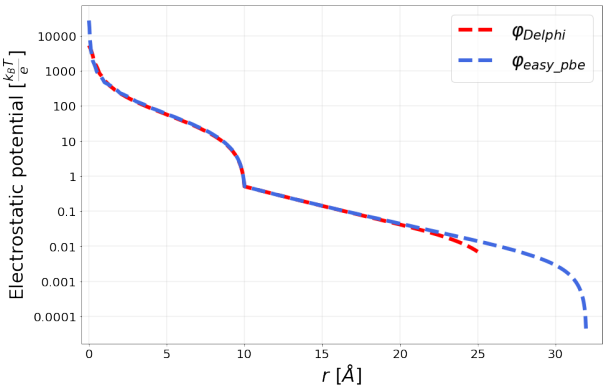
(b) Adaptive Oct-tree grid with maximum level of refinement equal to 11, minimum level equal to 6.

Figure 14: Exact solution (green) vs easy_pbe solution (blue).

For this problem, the comparison with Delphi is less accurate because the Coulombic boundary conditions for the PBE do not coincide with the exact ones, but they are based on the Debye–Hückel theory that are more precise with higher ionic strength values. We choose homogeneous Dirichlet boundary conditions for our code, which are the ones chosen for the simulations where the exact value is not known a priori. The potentials obtained from these numerical simulations are shown in Figure 15. The differences between the two curves are from about 20Åto 25Å, where the Ω_{Delphi} boundary is located. At 32Å, our solution drops to zero, as expected from the imposed BCs. The boundary, as expected, but the trends of the curves are very similar in the other points.



(a) Uniformly refined Oct-tree grid of level 8.



(b) Adaptive Oct-tree grid with maximum level of refinement equal to 11, minimum level equal to 6.

Figure 15: Delphi solution (red) vs easy_pbe solution (blue).

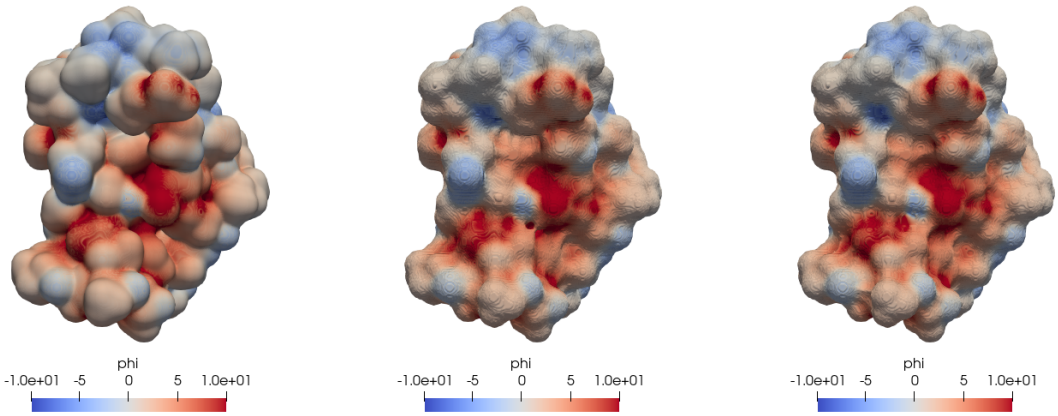
7 Numerical Results on Real Molecules

Nowadays, the use of molecular electrostatics simulations in chemistry and biology is still an important tool. The main applications regard the protein folding and structural-based drug design, which, by making use of cutting-edge HPC techniques, can be applied to large and complex molecules. We tested our code with different molecules for different targets, whose main results are presented in the following sections. The structural data of the biological macromolecules are all taken from RCBS Protein Data Bank [6].

7.1 Crambin Test Case

The first protein used for testing our code is a plant seed protein called crambin [8, 9]. It is relatively small as it is composed of 327 atoms and a unique protein chain. We employed the crambin molecule in order to both evidence the main differences between the three molecular surface definitions presented in this thesis and compare the potential with the one calculated by Delphi. The simulations have been performed on the computing server GIGAT of the MOX lab of Politecnico di Milano (6 nodes, 12 Intel Xeon E5-2640v4 @ 2.40GHz, 120 cores, 64GB

RAM per node, O.S. CentOS 7 interconnected by a dedicated Gigabit Ethernet), on 1 node and 8 processors. In Figure 16 you can see the electrostatic potential at the surface using respectively Blobby surface (Figure 16a), Skin surface (Figure 16b) and Connolly surface (Figure 16c). In the three examples, the domain extremes are the ones corresponding to the *stretched* mesh option, in this case $\Omega = [-22.73, 25.085] \times [-27.01, 27.146] \times [-22.169, 22.309]$.



(a) Blobby with B = -2.5. (b) Skin with s = 0.45. (c) SES with probe radius 1.4Å.

Figure 16: Electrostatic potential at the surface for a crambin molecule using different surface representations.

Looking at the three shapes, the Blobby one is most recognizable, while the Skin and SES surfaces, characterized by an higher livel of detail, are very similar to each other. This can be explained recalling that the Blobby surface definition is based on the level-set method, while Skin and SES rely both on alpha-shapes.
The electrostatic potential values at the molecular surface seem to be qualitatively similar in all the three cases. To inspect better the results, we plotted φ on a line passing through the center and parallel to the x axes, as shown in Figure 17. Notice that the line used for the extraction of the values is completely arbitrary and equivalent results can be obtained with other sampling choices. Also in this case the results given by the three surfaces are similar. The Skin and SES curves are extremely close one to the other, while the potential obtained with the Blobby surface presents some more evident discrepancies from the other two, especially inside the molecule. This is again attributable to the surface aspect, which affects the resulting grid and the numerical solution.

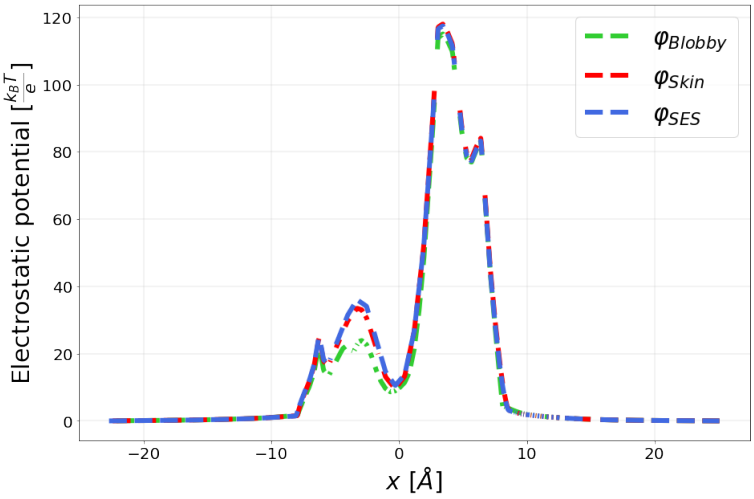


Figure 17: Electrostatic potential value for the crambin molecule over a line parallel to the x axes and passing through the center.

In table 2 we reported some useful information about the three simulations. The last two columns contain the number of cells corresponding to each test and the computational times for the principal operations made by the code:

- Computation of the electrostatic potential (P);
- Creation of the markers (M);
- Adaptive refinement of the mesh (R).

.

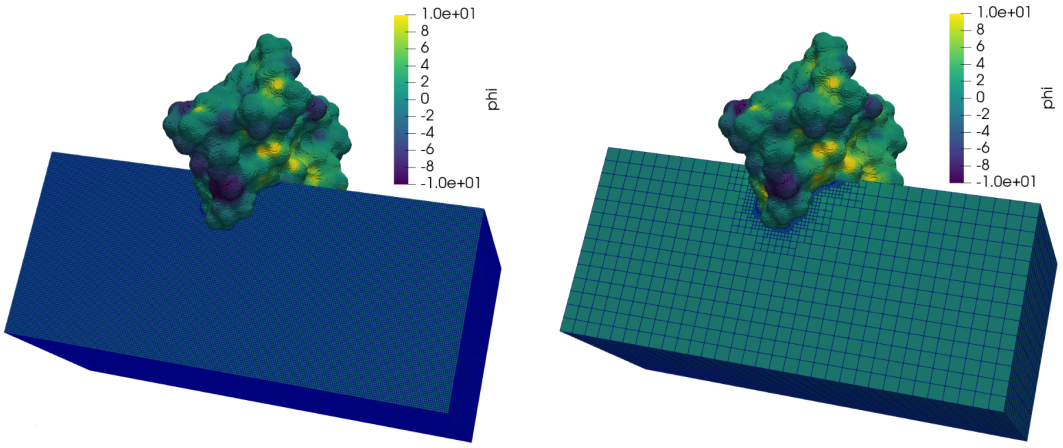
Both these aspects are comparable for the last two surfaces, while the Blobby one, which presents more elements, has the highest computational costs.

Surface type	min-max-uni level	Times	Number of mesh cells
Blobby	2-9-5	P: 47.3448 s M: 64.404 s R: 154.745 s	1334572
Skin	2-9-5	P: 44.529 s M: 5.750 s R: 62.073 s	1218722
SES	2-9-5	P: 45.913 s M: 5.668 s R: 61.610 s	1206458

Table 2: Comparison between different molecular surfaces for crambin molecule. Min-max-uni level refer to minumum, maximum and initial uniform level of refinement of the surface.

In order to obtain the same level of refinement at the surface, with a uniform mesh the number of elements would be higher than 134mln. This means that the adaptive refinement produces a saving of about 99,01%, 99,09% and 99,10% for Blobby, Skin and SES respectively.

In order to validate our solution, we compared the results obtained using the SES surface with the one given by Delphi solver. The Delphi potential was computed on a uniform grid, in which each side of the cubic domain was divided into 200 elements. Since our code can actually divide each side of the mesh by powers of two, we set the starting refinement level to 8, which corresponds to 256 subdivisions, and performed just one refinement iteration. Then, we projected the Delphi solution over the resulting grid, whose extremes were fixed equal to Delphi's ones. The first column in Figure 20 shows that the easy_pbe and Delphi potential values are almost exactly superposed.
Then, we tried to compare the Delphi results with a more pronounced refinement. In this case the minimum level was fixed to 5, while the maximum level to 9. In Figure 18 the differences between this and the previous mesh results.

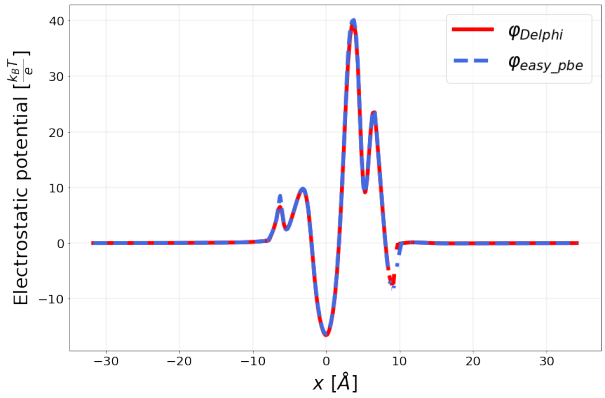


(a) minlevel = 8, maxlevel = 9.

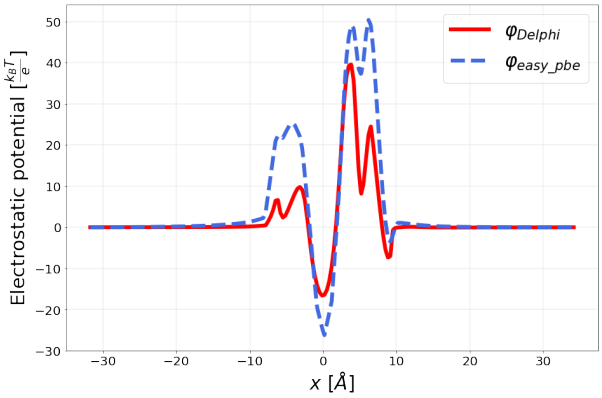
(b) minlevel = 5, maxlevel = 9.

Figure 18: Comparison between an almost uniform mesh and a refined one.

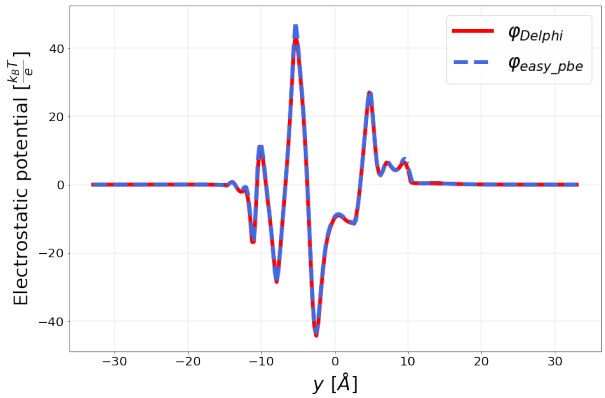
The accuracy is still acceptable, even if there are some evident different in the potential mostly inside the molecular surface, as shown from column two in Figure 20.



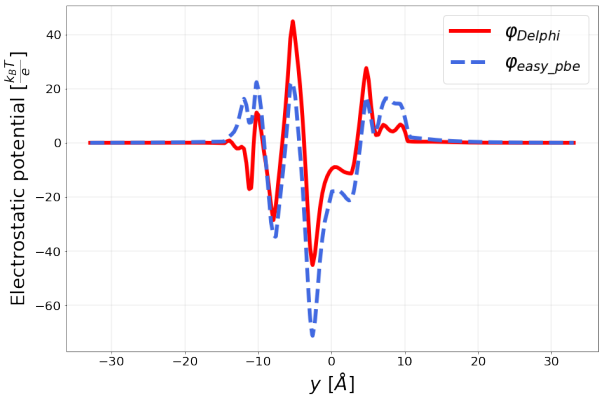
(a) Potential values on the line parallel to x axes, with minlevel = 8, maxlevel = 9.



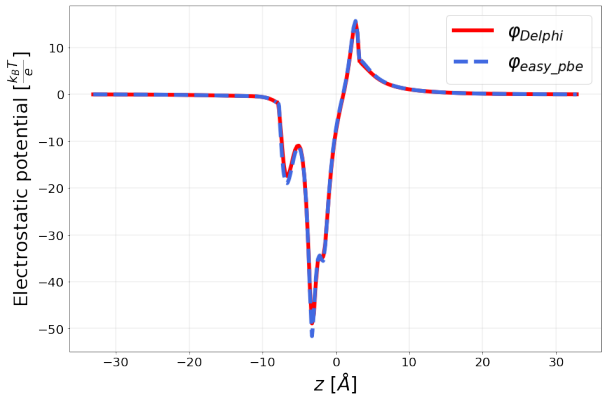
(b) Potential values on the line parallel to x axes, with minlevel = 5, maxlevel = 9.



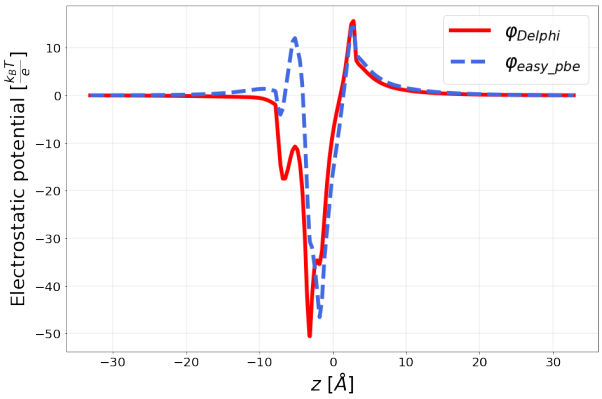
(c) Potential values on the line parallel to y axes, with minlevel = 8, maxlevel = 9.



(d) Potential values on the line parallel to y axes, with minlevel = 5, maxlevel = 9.



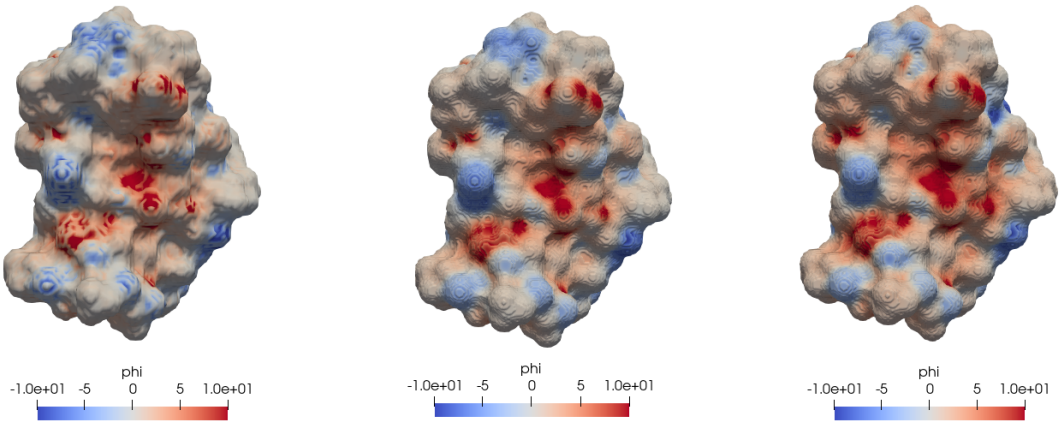
(e) Potential values on the line parallel to z axes, with minlevel = 8, maxlevel = 9.



(f) Potential values on the line parallel to z axes, with minlevel = 5, maxlevel = 9.

Figure 19: Delphi and easy_pbe potential values sampled on specific lines parallel to one axis and passing through the center of the domain.

Focusing on the potential values at the surface, shown in Figure 20, the results are comparable even if the last one is obtained with a mesh composed of about one tenth of cells of Delphi.



(a) Delphi solution.

(b) easy_pbe solution with parameters minlevel = 8, maxlevel = 9, surface = SES with 1.4Å probe radius.

(c) easy_pbe solution with parameters minlevel = 5, maxlevel = 9, surface = SES with 1.4Å probe radius.

Figure 20: Crambin electrostatic potential at the surface computed with Delphi and easy_pbe.

7.2 Parallel Scaling Test

In this section, we present the results of parallel scalability tests that have been performed again on the computing server GIGAT of the MOX lab of Politecnico di Milano. We choose the nerve growth factor, whose structure can be seen in Figure 21, which is an important protein that is involved in a variety of physiological processes in cell survival, differentiation, proliferation and maintenance [30, 31]. It is composed of a unique protein chain and 1750 atoms.

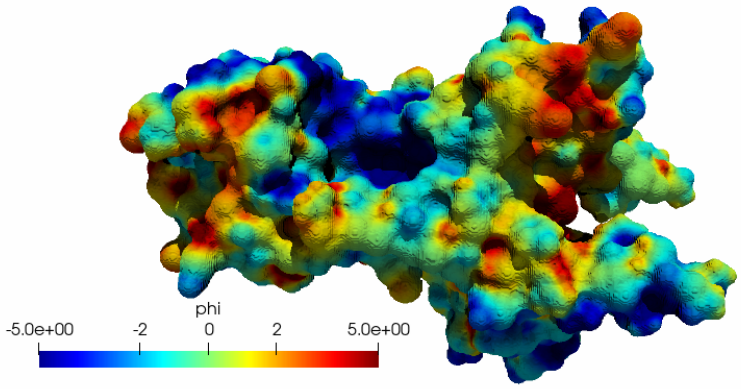
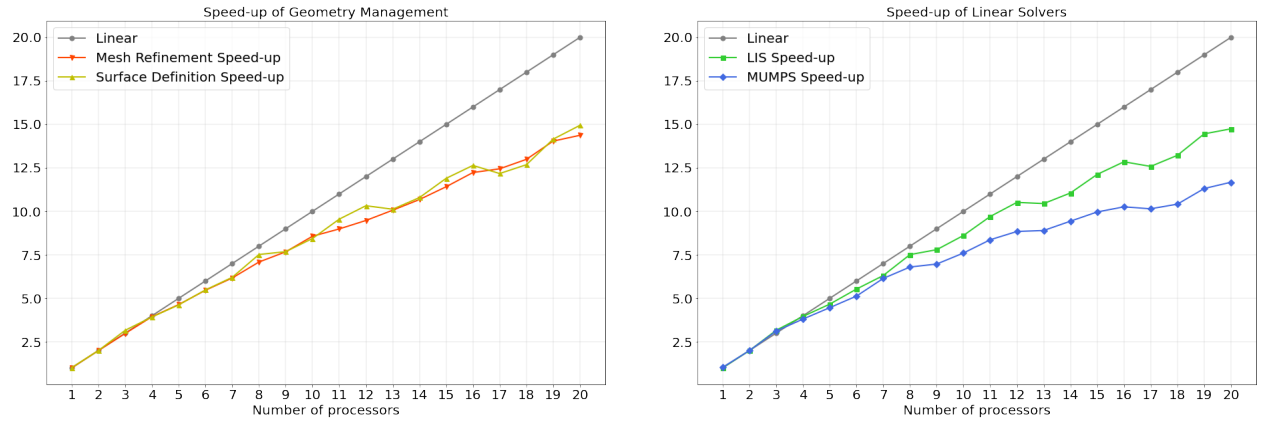


Figure 21: Electrostatic potential value calculated at the surface of crystal structure of Nerve growth factor.

The results of parallel scalability tests are reported in Figure 22. The trend of the curves is very close to the linear line up to 8 processors, after that, the distance with the linear curve increases, but the computational time saving is still satisfying. As expected, the direct solver is slower than the iterative one, as Figure 22b shows.



(a) (b)

Figure 22: Figures (a) and (b) refer respectively to the speed-up of the Geometry Management and Linear Solvers using 1 node and 1–20 processors.

8 Concluding Remarks and SARS-CoV-2 Spike Protein Simulation

The role of electrostatics in bio-molecular recognition is prominent, this is due to the features of electrostatic interactions both at long and short distances. Indeed, long range electrostatic interactions impact both on general repulsion/attraction of same and differently charged molecules, but also on the time that two biomolecules stay in close proximity, allowing them to find the most complementary conformations [19]. Short range electrostatics is also extremely important, since it helps achieve the specificity that is necessary for the correct functioning of biochemical signalling. Its contribution to solvation is the third element which makes electrostatics so relevant in this field.

The Poisson Boltzmann equation is routinely used to estimate the role of electrostatics and desolvation to molecular binding, as well as to pKa shifts in titratable residues and in many other situations. It proved able to pinpoint molecular determinants of protein-protein interaction specificity [29] as well as the effects of shape in protein-DNA interaction [28].

For such reasons, Poisson-Boltzmann calculations, may deserve to be considered one of the enabling technologies for developing automatic strategies for drug discovery or drug repurposing.

To test the applicability of our solver on molecular structures of actual interest for applications we ran `easy_pbe` simulations on two molecules that are clearly relevant targets for this sort of studies, in particular the spike protein of two different strains of the SARS-CoV-2 virus, e first strain [33, 34] versus the Omicron variant [37, 38]. Results reported in Figure 23 were obtained using SES surface and took about 4 and 3 hours respectively on the computing server GIGATLONG of the MOX lab of Politecnico di Milano, using 20 processors. The times are divided in the following way:

- Between 59-60% for the refinement and coarsening steps,
- between 39-40% for the calculus of the electrostatic potential,
- the remaining time for the markers computation and the export of the calculated quantities, mesh included.

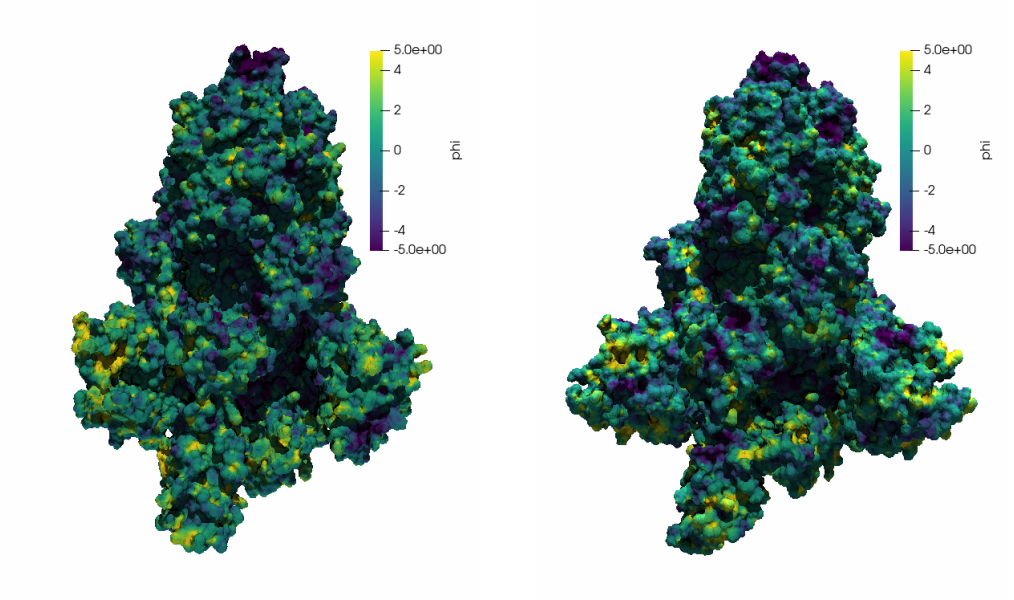


Figure 23: SARS-CoV-2, spike protein, ectodomain structure open state (left), Omicron variant of concern (right).

References

[1] `https://github.com/carlodefalco/easy_pbe.git`.

[2] `http://p4est.github.io`.

[3] `http://optimad.github.io/bitpit/modules/index.html#PABLO`.

[4] `https://www.ssisc.org/lis/lis-ug-en.pdf`.

[5] N. Baker, M. Holst, and F. Wang. Adaptive multilevel finite element solution of the poisson-boltzmann equation ii. refinement at solvent-accessible surfaces in biomolecular systems. *J. Comput. Chem.*, 21:1343–1352.

[6] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000. `http://www.rcsb.org/`.

[7] J.F. Blinn. A generalization of algebraic surface drawing. *ACM T. Graph.*, 1(3):235–256, 1982.

[8] A.M. Bonvin, J.A. Rullmann, R.M. Lamerichs, R. Boelens, and R. Kaptein. Direct noe refinement of crambin from 2d nmr data using a slow-cooling annealing protocol, 1993. PDB ID: 1CCM, PDB doi: `http://doi.org/10.2210/pdb1CCM/pdb`.

[9] A.M. Bonvin, J.A. Rullmann, R.M. Lamerichs, R. Boelens, and R. Kaptein. Ensemble iterative relaxation matrix approach: a new nmr refinement protocol applied to the solution structure of crambin. *Proteins*, 15:385–400, 1993. doi: `http://dx.doi.org/10.1002/prot.340150406`.

[10] M. Born. Volumen und hydratationswärme der ionen. *Z. Phys.*, 1(45), 1920.

[11] A. H. Boschitsch and M. O. Fenley. A fast and robust poisson–boltzmann solver based on adaptive cartesian grids. *Journal of Chemical Theory and Computation*, 7(5):524–1540, 05 2011.

[12] S. Decherchi, José C., C. E. Catalano, M. Spagnuolo, E. Alexov, and W. Rocchia. Between algorithm and model: different molecular surface definitions for the poisson-boltzmann based electrostatic characterization of biomolecules in solution. *Commun Comput Phys.*, 13:61–89, January 2013.

[13] S. Decherchi, C. E. Catalano J. Colmenare and, M. Spagnuolo, E. Alexov, and W. Rocchia. Between algorithm and model: different molecular surface definitions for the poisson-boltzmann based electrostatic characterization of biomolecules in solution. *Commun. Comput. Phys.*, 13:61–89, January 2013.

[14] S. Decherchi and W. Rocchia. A general and robust ray-casting-based algorithm for triangulating surfaces at the nanoscale. *PLoS One*, 8(4):e59744, 2013. http://dx.doi.org/10.1371/journal.pone.0059744.

[15] H. Edelsbrunner. Deformable smooth surface design. *Disrete Comput. Geom.*, 21:87–115, 1999.

[16] E. Jurrus et al. Improvements to the APBS biomolecular solvation software suite. *Protein Science*, 27:112–128, 2018.

[17] M. Holst, N. Baker, and F. Wang. Adaptive multilevel finite element solution of the poisson–boltzmann equation i. algorithms and examples. *J. Comput. Chem.*, 21:1319–1342.

[18] B. Honig and A. Nicholls. Classical electrostatics in biology and chemistry. *SCIENCE*, 268, MAY 1995.

[19] M. Kosloff, A. Travis, and D. Bosch et al. Integrating energy calculations with functional assays to decipher the specificity of g protein–rgs protein interactions. *Nat Struct Mol Biol*, 18:846—-853, 2011.

[20] A. Kucherova, S. Strango, S. Sukenik, and M. Theillard. Computational modeling of protein conformational changes - application to the opening sars-cov-2 spike. *Journal of Computational Physics*, 444:110591, 2021.

[21] A. Lee, W. Geng, and S. Zhao. Regularization methods for the poisson-boltzmann equation: Comparison and accuracy recovery. *Journal of Computational Physics*, 426:109958, 2021.

[22] B. Lee and F. M. Richards. The interpretation of protein structures: Estimation of static accessibility. *J. Mol. Biol.*, 55:379–400, 1971.

[23] C. Li, Z. Jia, A. Chakravorty, S. Pahari, Y. Peng, S. Basu, M. Koirala, S.K. Panday, M. Petukh, L. Li, and E. Alexov. Delphi suite: New developments and review of functionalities. *J Comput Chem.*, 40(28):2502–2508, Oct 2019.

[24] L. Li, C. Li, S. Sarkar, J. Zhang, S. Witham, Z. Zhang, L. Wang, N. Smith, M. Petukh, and E. Alexov. Delphi: a comprehensive suite for delphi software and associated resources. *BMC Biophys*, 4(1), May 2012.

[25] M. Mirzadeh, M. Theillard, and F. Gibou. A second-order discretization of the nonlinear poisson–boltzmann equation over irregular geometries using non-graded adaptive cartesian grids. *Journal of Computational Physics*, 230(5):2125–2140, 2011.

[26] A. Nicholls and B. Honig. A rapid finite difference algorithm, utilizing successive over-relaxation to solve the poisson-boltzmann equation. *J. Comput. Chem.*, 12:435–445., 1991.

[27] W. Rocchia. Poisson-boltzmann equation boundary conditions for biological applications. *Mathematical and Computer Modelling*, 41(10):1109–1118, 2005. Modelling Complex Systems in Molecular Biology and Tumor Dynamics and Control.

[28] R. Rohs, S. M. West, A. Sosinsky, P. Liu, R. S. Mann, and B. Honig. The role of dna shape in protein – dna recognition. *nature*, 461:1248–1253, 2009.

[29] F. B. Sheinerman, R. Norel, and B. Honig. Electrostatic aspects of protein–protein interactions. *Current Opinion in Structural Biology*, 10(2):153–159, 2000.

[30] H.L. Sun and T. Jiang. Crystal structure of nerve growth factor in complex with lysophosphatidylinositol, 2015. PDB ID: 4XPJ, PDB doi: http://doi.org/10.2210/pdb4XPJ/pdb.

[31] H.L. Sun and T. Jiang. The structure of nerve growth factor in complex with lysophosphatidylinositol. *Acta Crystallogr F Struct Biol Commun*, 71:906–912, 2015. doi: http://dx.doi.org/10.1107/S2053230X15008870.

[32] E. Alexov W. Rocchia and B. Honig. Extending the applicability of the nonlinear poisson-boltzmann equation: Multiple dielectric constants and multivalent ions. *J. Phys. Chem. B*, 105(28):6507–6514.

[33] A.C. Walls, Y.J. Park, M.A. Tortorici, A. Wall, Seattle Structural Genomics Center for Infectious Disease (SSGCID), A.T. McGuire, and D. Veesler. Sars-cov-2 spike ectodomain structure (open state), 2020. PDB ID: 6VYB, PDB doi: http://doi.org/10.2210/pdb6VYB/pdb.

[34] A.C. Walls, Y.J. Park, M.A. Tortorici, A. Wall, A.T. McGuire, and D. Veesler. The structure of nerve growth factor in complex with lysophosphatidylinositol. *Cell*, 181:281, 2020. doi: http://dx.doi.org/10.1016/j.cell.2020.02.058.

[35] J. A. Wilson, A. Bender, T. Kaya, and P. A. Clemons. Alpha shapes applied to molecular shape characterization exhibit novel properties compared to established shape descriptors. *J. Chem. Inf. Model.*, 49(10):2231–2241, 2009.

[36] Y. Zhang, G. Xu, and C. Bajaj. Quality meshing of implicit solvation models of biomolecular structures. *Comput. Aided Geom. Des.*, 23:510–530, 2006.

[37] T. Zhou, T. Tsybovsky, and P.D. Kwong et al. Antibodies with potent and broad neutralizing activity against antigenically diverse and highly transmissible sars-cov-2 variants. *Biorxiv*, 2021. doi: `http://dx.doi.org/10.1101/2021.02.25.432969`.

[38] T. Zhou, T. Tsybovsky, and P.D. Kwong. Cryo-em structure of the spike of sars-cov-2 omicron variant of concern, 2021. PDB ID: 7TB4, PDB doi: `http://doi.org/10.2210/pdb7TB4/pdb`.

A `easy_pbe` user reference

This appendix contains the instructions on how to compile and use the code and visualize the outputs.
The implemented code can be found inside the repository: `https://github.com/carlodefalco/easy_pbe`, and its structure is schematically reported in Figure 24.

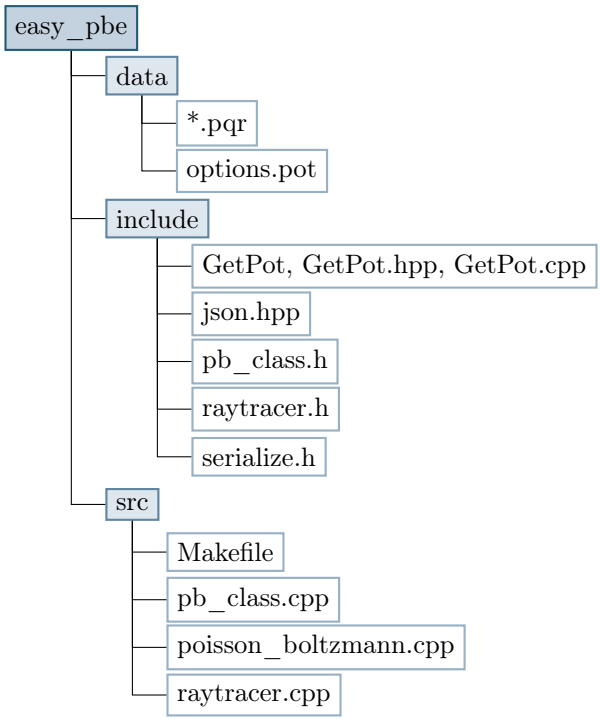


Figure 24: Repository structure scheme with both folders and files.

The repository contains three directories. Inside the folder `data` two kind of files are included: the `option.pot` file, in which the user can set the parameters for the simulation, and some `.pqr` files with the molecule information. The `include` and `src` directories respectively contain the header and source files of the code. Inside the `src` folder the main function `poisson_boltzmann.cpp` and the `Makefile`, necessary for the compilation, can be found.

A.1 Installation

The direct dependencies of the code are `bim++` and `NanoShaper`, while Paraview and `octave` are necessary for the post-processing operations.
For the compilation, a `local_settings.mk` file must be created in order to add or modify, accordingly to the user necessities, the compilation options present inside `Makefile`, and then the following command must be executed

```
make
```

A.2 Input files

The code takes as input two files:

- `options.pot` file, containing the parameters for the simulation;
- .pqr file.

The .pqr file format is a modification of the .pdb format, which is used for the description of the 3D structure of the molecules of the Protein Data Bank. The correct .pqr format can be generated starting from the .pdb format using the PDB2PQR tool (`https://server.poissonboltzmann.org/pdb2pqr`) and selecting TYL06 as output naming scheme with the *whitespaces* option activated together with the default ones. Inside a .pqr file all the information about the studied molecules can be found. In Figure 25 the first lines of the crambin molecule .pqr file are reported as an example of its content.

fieldname	Atom_number	Atom_name	Residue_name	Residue_number	X	Y	Z	Charge	Radius
ATOM	1	N	THR	1	-4.764	-6.422	3.592	0.1812	1.824
ATOM	2	CA	THR	1	-3.319	-6.147	3.458	0.0034	1.908
ATOM	3	C	THR	1	-2.898	-5.41	2.18	0.6163	1.908
ATOM	4	O	THR	1	-3.23	-5.781	1.062	-0.5722	1.6612
ATOM	5	CB	THR	1	-2.521	-7.479	3.58	0.4514	1.908
ATOM	6	OG1	THR	1	-2.908	-8.098	4.811	-0.6764	1.721
ATOM	7	CG2	THR	1	-0.982	-7.503	3.488	-0.2554	1.908
ATOM	8	H1	THR	1	-5.267	-5.559	3.62	0.1934	0.6
ATOM	9	HA	THR	1	-3.034	-5.578	4.26	0.1087	1.1
ATOM	10	HB	THR	1	-2.789	-8.083	2.816	-0.0323	1.387
ATOM	11	HG21	THR	1	-0.659	-8.44	3.614	0.0627	1.487
ATOM	12	HG22	THR	1	-0.7	-7.168	2.59	0.0627	1.487
ATOM	13	HG23	THR	1	-0.6	-6.916	4.201	0.0627	1.487

Figure 25: First lines of 1CCM.pqr file.

The `fieldname` corresponds to a string which specifies the type of the PQR entry, it can be either ATOM or HETATM. The `Atom_number` and `Atom_name` are respectively an integer with the index of each atom and a string containing its name, while the following two fields regards the residue. The remaining columns are the most important because they contain the information for the electrostatic potential computation: the atomic coordinates of the atom center (expressed in Å), the charge (expressed in electrons unit) and the atom radius (expressed in Å).

An example of option file can be seen in Figure 26.

```
#Parameters file

[mesh]
minlevel = 2
maxlevel = 9
unilevel = 5
mesh_shape = 1
x1 = -30.424
x2 = 33.576
y1 = -31.92
y2 = 32.08
z1 = -31.9
z2 = 32.1
[../]

[model]
linearized = 1
bc_type = 1
molecular_dielectric_constant = 2
solvent_dielectric_constant = 80
ionic_strength = 0.145
[../]

[surface]
surface_type = 0
surface_parameter = 1.4
stern_layer_thickness = 2.
number_of_threads = 1
[../]

[algorithm]
linear_solver = lis
solver_options = -i\ cg\ -p\ ilu\ -ilu_fill\ [0]\ -tol\ 1.e-12\
[../]

[output]
p4estfilename = poisson_boltzmann_p4est
markerfilename = poisson_boltzmann_marker_0
surffilename = poisson_boltzmann_surface_0
[../]
```

Figure 26: Example of `options.pot` file.

Below the description of the parameters is reported.
Mesh options:

- maxlevel: maximum level for the mesh refinement, root assumed level 0.
- minlevel: minimum level for the mesh refinement.
- unilevel: level chosen for the initial uniform refinement of the mesh.
- mesh shape: it can be *cubic* (option 0), *stretched* (option 1) or *manually set* (option 2). Consider for instance the problem domain

$$\Omega = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$$

With *cubic* option, the list of all the atoms inside the examined molecule is scrolled in order to find the maximum (minimum) coordinate value without distinguishing among the three axis directions. Then the right (left) domain extremes of each interval in Ω are set equal to this value summed (subtracted) to six times the maximum radius among all the atoms of the chosen molecule. In this way the min and max values of each interval are the same along the coordinates.
For the *stretched* option instead, the maximum (minimum) coordinate value is searched separately in every direction, so that in general $x_{min} \neq y_{min} \neq z_{min}$ and $x_{max} \neq y_{max} \neq z_{max}$.
The user can also decide to set the domain coordinates explicitly by selecting the *manual setting* and filling the corresponding spaces with the desired domain vertex coordinates.

Model parameters:

- linearized: 1 for solving the *linearized* PBE, the only available up to now.
- bc_type: 0 for homogeneous Neumann, 1 for homogeneous Dirichlet.
- molecular_dielectric_constant: relative permittivity constant of the molecule.
- solvent_dielectric_constant: relative permittivity constant of the solvent.
- ionic_strength: I expressed in molar concentration.

`NanoShaper` surface parameters:

- surface_type:
  - 0 for SES (or Connolly surface);
  - 1 for Skin surface;
  - 2 for Blobby surface.
- surface_parameter:
  - SES probe radius (default value 1.4Å).
  - Skin surface parameter $s \in (0, 1)$. The default value is 0.45, which corresponds to a surface very similar to the SES one. If $s = 1$, it corresponds to the convex hull, if $s = 0$ it coincides with the Van der Waals surface.
  - Blobby parameter B (default value -2.5, not too far from SES).
- stern_layer_thickness: (not yet available for `NanoShaper` API).
- number_of_threads: set the available threads for `NanoShaper` surface computation.

Algorithm options:

- linear_solver: the user can decide to use either `mumps` direct solver, or `lis` iterative solver.
- solver_options: only for the iterative solver, see `lis` manual [4] for the complete list of available options, *e.g.* solvers, preconditioners, tolerance, etc.

Output options: arbitrary names for the creation of the output files.

A.3 Launching Simulations from the Command Line

To run the code in serial, using the default `options.pot` file and the 1CCM.pqr file, the necessary commands are

```
mkdir results
cd results
../poisson_boltzmann
```

To run in parallel with MPI, change the last line with

```
mpirun -np <number of processes> ../poisson_boltzmann
```

with `processors_number` equal to the desired number of processors.
To select a different option file add the following option

```
--potfile /your/folder/<file name>.pot
```

While to change the input molecule add the following option

```
--pqrfile /your/folder/<file name>.pqr
```

A.4 Post-processing of Output Files

The user can use the `octave` script called `export.m` and distributed with `bimpp` (it is located in the folder `script/m`) for the post-processing. When the user adopts the default names for the files, it can write the following command line for the creation of a .vtu file that can be opened using Paraview.

```
export_tmesh_data ('poisson_boltzmann_surface_%1.1d_%4.4d',
                   {'poisson_boltzmann_surface_%1.1d_%4.4d', 'phi_%1.1d_%4.4d',
                   'rhs_%1.1d_%4.4d', 'rho_%1.1d_%4.4d'}, {'surface', 'phi', 'rhs', 'rho'},
                   {'poisson_boltzmann_marker_%1.1d_%4.4d', 'epsilon_%1.1d_%4.4d',
                   'reaction_%1.1d_%4.4d'}, {'marker', 'epsilon', 'reaction'},
                   'output_name', 0, processors_number)
```

where `output_name` can be substituted with an arbitrary name for the .vtu file and `processors_number` must be replaced by the number of processors used for the simulation. The user can also decide to convert only some among the outputs according to his necessities.

Abstract in lingua italiana

Per capire le proprietà fisiche e chimiche delle molecole immerse in soluzioni acquose, è fondamentale una descrizione accurata della forza e della natura delle interazioni elettrostatiche, le quali giocano un ruolo fondamentale in una grande quantità di processi biologici. L'utilizzo dei cosiddetti *continuum solvent models*, come l'equazione di Poisson–Boltzammn (PBE), per la modellizzazione di queste interazioni in sistemi di biomolecole di una sempre meggior complessità, dipende fortemente dalla possibilità di utilizzare e implementare metodi numerici efficienti, accurati e scalabili, in modo da sfruttare l'elevato parallelismo delle architetture dei computer moderni. Questa tesi si pone in questo contesto con l'implementazione di `easy_pbe`, un solutore numerico scalabile per l'equazione linearizzata di Poisson–Boltzmann. Il solver è basato sulla discretizzazione a Elementi Finiti della PBE su mesh cartesiane gerarchicamente raffinate e permette di scegliere tra diversi approcci per la descrizione della superficie molecolare. Dopo aver introdotto la formulazione del problema, e la sua forma adimensionale, verranno descritti nel dettaglio alcuni aspetti inerenti al solutore implementato, come l'approccio utilizzato per trattare le singolarità causate dalla presenza delle cariche puntiformi all'interno del modello. Successivamente verranno presentati alcuni casi test significiativi per la validazione dell'accuratezza del codice e la sua scalabilità in parallelo.

Parole chiave: Elettrostatica molecolare, equazione di Poisson–Bolztmann, superficie molecolare, mesh Oct-tree, parallelizzazione MPI.

Acknowledgements

First of all I would like to thank my advisor professor Carlo de Falco for the patience, the precious advices and all the aid he gave me during all the period we worked on this thesis. Then, I kindly acknowledge dr. Walter Rocchia and dr. Sergio Decherchi for their support, the useful discussions and their genuine interest.

Ringrazio di cuore la mia famiglia, mia mamma, mio papà e Andrea, senza il cui supporto non avrei potuto ottenere neanche la metà delle piccole e grandi soddisfazioni raggiunte durante il mio percorso di studi, e non solo. Ringrazio i miei amici, quelli di una vita, Alessia e Alessandro, che ci sono sempre stati e che mi hanno sempre ascoltata. Ringrazio anche gli amici che si sono aggiunti in questi anni, in particolare Alessia, con cui ho stretto un rapporto molto speciale, e Sharon, con cui ho condiviso le gioie e le difficoltà incontrate al Politecnico e senza la quale sarebbe stato tutto più difficile. Infine ringrazio di cuore Davide, per le notti insonni passate insieme ad aiutarci, senza mai tirarsi indietro, per aver creduto in me anche quando ero io la prima a non farlo e per tutto l'amore dimostratomi.