How to make continuum solvation incredibly fast in a few simple steps: a practical guide to the domain decomposition paradigm for the Conductor-like Screening Model Continuum Solvation, Linear Scaling, Domain Decomposition

Benjamin Stamm, Louis Lagardère, Giovanni Scalmani, Paolo Gatto, Eric Cancès, Jean-Philip Piquemal, Yvon Maday, Benedetta Mennucci, Filippo Lipparini

▶ **To cite this version:**

Benjamin Stamm, Louis Lagardère, Giovanni Scalmani, Paolo Gatto, Eric Cancès, et al.. How to make continuum solvation incredibly fast in a few simple steps: a practical guide to the domain decomposition paradigm for the Conductor-like Screening Model Continuum Solvation, Linear Scaling, Domain Decomposition. International Journal of Quantum Chemistry, 2019, 119, pp.e25669. 10.1002/qua.25669 . hal-01793770

HAL Id: hal-01793770
https://hal.science/hal-01793770

Submitted on 16 May 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SOFTWARE NEWS

How to make continuum solvation incredibly fast in a few simple steps: a practical guide to the domain decomposition paradigm for the Conductor-like Screening Model

Benjamin Stamm[1] | **Louis Lagardère**[2,3,4] | **Giovanni Scalmani**[5] | **Paolo Gatto**[6] | **Eric Cancès**[7] | **Jean-Philip Piquemal**[3,4,9] | **Yvon Maday**[8,9,11] | **Benedetta Mennucci**[12] | **Filippo Lipparini**[12]*

[1]Mathematics Division, Center for Computational Engineering Science, RWTH Aachen University,Aachen, Germany

[2]Sorbonne Universite, Institut des Sciences du Calcul et des Donnees, Paris, France

[3]Sorbonne Universite, Institut Parisien de Chimie Physique et Theorique, FR 2622 CNRS, Paris, France

[4]Sorbonne Universite, Laboratoire de Chimie Theorique, UMR 7616 CNRS, Paris, France

[5]Gaussian, inc. 340 Quinnipiac St., Building 40, Wallingford, CT 06492, United States

[6]Aachen Institute for Advanced Study in Computational Engineering Science (AICES), RWTH Aachen University, Aachen, Germany

[7]CERMICS, École des Ponts and INRIA, Université Paris-Est, 6 & 8 avenue Blaise Pascal, 77455 Marne-la-Vallée, France

[8]Sorbonne Universite, Laboratoire Jacques-Louis Lions, UMR 7598 CNRS, Paris, France

[9]Institut Universitaire de France, Paris, France

[10]Department of Biomedical Engineering, The University of Texas at Austin, Austin, Texas 78712, United States

[11]Brown University, Division of Applied Maths, Providence, RI, USA

[12]Dipartimento di Chimica e Chimica

We illustrate the domain decomposition Conductor-like Screening Model (ddCOSMO) implementation and how to couple it with an existing classical or quantum mechanical (QM) code. We review in detail what input needs to be provided to ddCOSMO and how to assemble it, describe how the ddCOSMO equations are solved and how to process the results in order to assemble the required quantities, such as Fock matrix contributions for the QM case, or forces for the classical one. Throughout the paper, we will make explicit references to the ddCOSMO module, which is an open source, Fortran 90 implementation of ddCOSMO that can be downloaded and distributed under the LGPL license.

KEYWORDS
Continuum Solvation, Linear Scaling, Domain Decomposition

1 | INTRODUCTION

Continuum Solvation models (CSMs) [1, 2, 3, 4, 5] are nowadays part of the standard toolbox of computational chemists. Historically, in the quantum chemistry community, polarizable CSMs such as the Polarizable Continuum Model (PCM) [1] or the Conductor-like Screening Model (COSMO) [6] have been developed as a cheap, but physically sound way, to include solvation effects in the quantum mechanical (QM) description of a molecule and its properties. As the computational cost is usually dominated by the solution of the QM equations, the computational performance of CSM have not been historically taken into much consideration, as the setup and solution of the CSM equations, which are equivalent in some way to solving Poisson's equation in a heterogeneous dielectric medium, has always been assumed to be a negligible additional computational cost. Due to advances in hardware, more efficient implementations and the spread of linear scaling techniques within quantum chemistry, such an assumption started to be less and less true in the last decade. Further, the diffusion of multiscale methods such as quantum mechanics/molecular mechanics (QM/MM) has made large to very large systems accessible to computational chemists. For such systems, the computational cost associated with continuum solvation, which scales as the second or third power of the size of the system, can easily become the real bottleneck of the calculation [7]. Nevertheless, polarizable CSM have been used successfully in a large number of different applications and are a standard feature of most quantum chemistry codes. Polarizable CSM have not had the same success in Molecular Dynamics (MD) applications, as their computational cost makes them incompatible with the usual time frame of these simulations. For this reason, the CSMs used in classical MD are usually additive approximation to Poisson's equation such as the Generalized Born model [8].

Recently, a new paradigm has been introduced for the solution of the COSMO equations [9, 10]. The new algorithm, called ddCOSMO, is based on domain decomposition (dd) and exhibits linear scaling and an overall very limited computational cost [10, 7, 11]. With respect to existing implementations of CSM [12, 7], ddCOSMO is two to three orders of magnitude faster, effectively allowing the use of a polarizable CSM for large systems. Furthermore, the ddCOSMO discretization is systematically improvable and controlled by a very limited number of parameters, making it easy to control and overall rigorous and sound[9]. It also provides for a smooth energy as a function of the nuclear coordinates, making it suitable for geometry optimization and molecular dynamics simulation[13, 7]. However, its implementation is slightly more cumbersome than standard CSM implementations based on the boundary element method (BEM)[14] or the York-Karplus (YK) method[15, 16, 17]. A stand-alone, opensource, modular implementation of ddCOSMO is distributed for free under the terms of the version 3 of the GNU Lesser General Public Licence (LGPL) license. It can be download from `https://www.ddpcm.org`, or directly from GitHub[18].

The goal of this communication is to provide the essential knowledge to successfully use and couple the stand-alone library to the code of choice, or implement the method from scratch. We provide all necessary technical details and review the implementation of ddCOSMO and its coupling with an existing QM or classical code in this communication.

This paper is organized as follows. In section 2, the main aspects of the ddCOSMO paradigm are succinctly presented. In section 3, we will discuss the coupling of ddCOSMO with an existing code and review what are the quantities that the existing code needs to assemble in order to compute the self-consistent field energy for a QM code and the energy and forces for a classical code. In section 4, the implementation of ddCOSMO is described with explicit references to the open source ddCOSMO module. Finally, we will provide some conclusion and perspectives in section 5.

2 | A BRIEF INTRODUCTION TO THE ddCOSMO PARADIGM

The idea of this section is to give a different derivation of the ddCOSMO equations than what was presented in the original articles [9, 10, 11, 19] that might be more intuitive to follow. In particular, we refer to the appendix B of [20] which presents an alternative derivation of the ddCOSMO equations. We start with recalling the Partial Differential Equation (PDE) associated to the COSMO (model). The reaction potential W satisfies

$$\begin{aligned} -\Delta W &= 0, && \text{in } \Omega = \Omega_1 \cup \cdots \cup \Omega_M, \\ W &= -\Phi, && \text{on } \partial\Omega, \end{aligned}$$

where Δ denotes the Laplacian, Φ the free-space potential generated by the solute's charge distribution ρ and Ω is the solute's cavity consisting of a union of, possibly scaled, Van der Waals balls Ω_i, each centered at x_i and of radius r_i.

The idea of ddCOSMO is to rewrite this problem into M coupled problems:

$$-\Delta W_i = 0, \qquad \text{in } \Omega_i, \tag{1}$$

$$W_i = -\Phi, \qquad \text{on } \partial\Omega_i \cap \partial\Omega, \tag{2}$$

$$W_i = \textstyle\sum_{j\in\mathcal{N}_i} \omega_{ij} W_j, \qquad \text{on } \partial\Omega_i \cap \Omega, \tag{3}$$

where the right hand side of (3) at some point $x \in \partial\Omega_i$ denotes the average value of all neighboring potentials $W_j(x)$ corresponding to all neighboring spheres Ω_j intersecting Ω_i at x, i.e. for all $j \neq i$ such that $x \in \Omega_j$. More precisely, we denote by $\mathcal{N}_i$ the set of all spheres intersecting sphere i and the function ω_{ij} denotes the weight function defined by

$$\forall x \in \partial\Omega_i : \qquad \omega_{ij}(x) = \frac{\chi_j(x)}{\sum_{k\in\mathcal{N}_i} \chi_k(x)},$$

using the convention that $0/0 = 0$ and where χ_j is the characteristics function of Ω_j. Therefore, $\omega_{ij}(x)$ can take the values 0 (exposed to the solvent at x), 1 (one intersecting sphere at x), $1/2$ (two intersecting spheres at x), $1/3$, $1/4$ etc. Following this construction, we deduce that $U_i(x) = 1 - \sum_{j\in\mathcal{N}_i} \omega_{ij}(x)$ denotes the characteristics function of the part of $\partial\Omega_i$ belonging to $\partial\Omega$ taking only values 0 (inside Ω) or 1 (on $\partial\Omega$).

On the discrete level, we represent the local approximations W_i to W in each Ω_i by

$$x \in \Omega_i : \qquad W|_{\Omega_i}(x) \approx W_i(x) = \sum_{\ell m} \frac{4\pi}{2\ell+1} [X_i]_\ell^m \, r_\ell^i(x) \, Y_{\ell,m}^i(x), \tag{4}$$

with radial scaling $r_\ell^i(x) := \left(\frac{|x-x_i|}{r_i}\right)^\ell$ and angular dependency $Y_{\ell,m}^i(x) := Y_\ell^m\left(\frac{x-x_i}{|x-x_i|}\right)$ relative to the i-th atom. The symbol $\sum_{\ell m}$ denotes $\sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell}$ representing a truncated series of real spherical harmonics Y_ℓ^m. Since W_i is by construction harmonic in Ω_i, we only need to match the boundary conditions on each $\partial\Omega_i$, which is done by multiplying (2)–(3) on any $\partial\Omega_i$ by any spherical harmonics $Y_{\ell,m}^i$ up to degree $\ell_{\max}$ and integrating over $\partial\Omega_i$. However, since exact integration cannot be carried out, we employed a Lebedev quadrature[21] given by weights $\omega_n \in \mathbb{R}$ and integration points $s_n \in \mathbb{S}^2$ on the unit sphere to provide a (scaled) numerical integration scheme on each $\partial\Omega_i$:

$$\langle f, g \rangle_{n,i} := \sum_{n=1}^{N_g} \omega_n \, f(x_n^i) \, g(x_n^i) \approx \frac{1}{r_i^2} \int_{\partial\Omega_i} f(s) g(s) ds = \int_{\mathbb{S}^2} f(x_i + r_i\hat{s}) g(x_i + r_i\hat{s}) d\hat{s},$$

with $x_n^i := x_i + r_i s_n$. Then, we require the following equations to hold for all $i = 1, \dots, M$, $\ell = 0, \dots, \ell_{\max}$ and $|m| \leq \ell$:

$$\left\langle W_i, Y^i_{\ell,m} \right\rangle_{n,i} = \left\langle \sum_{j \in \mathcal{N}_i} \omega_{ij} W_j - U_i \Phi, Y^i_{\ell,m} \right\rangle_{n,i}. \tag{5}$$

This results in a linear system

$$LX = g \tag{6}$$

which is naturally block-sparse since only intersecting spheres provide non-zero block entries. While X is just a vector containing the coefficients $[X_i]^m_\ell$, ordered block-wise for all spheres i, the matrix and right hand side coefficients are given by (for all $i = 1, \dots, M, j \in \mathcal{N}_i$):

$$[L_{ii}]^{mm'}_{\ell\ell'} = \tfrac{4\pi}{2\ell'+1} \left\langle Y^i_{\ell',m'}, Y^i_{\ell,m} \right\rangle_{n,i} = \tfrac{4\pi}{2\ell'+1} \delta_{\ell\ell'} \delta_{mm'}, \tag{7}$$

$$[L_{ij}]^{mm'}_{\ell\ell'} = -\tfrac{4\pi}{2\ell'+1} \left\langle \omega_{ij} r^j_{\ell'} Y^j_{\ell',m'}, Y^i_{\ell,m} \right\rangle_{n,i} = -\tfrac{4\pi}{2\ell'+1} \sum_{n=1}^{N_g} \omega_n \, \omega_{ij}(x_n^i) \, r^j_{\ell'}(x_n^i) \, Y^j_{\ell',m'}(x_n^i) \, Y^m_\ell(s_n), \tag{8}$$

$$[g_i]^m_{\ell'} = -\left\langle U_i \Phi, Y^i_{\ell,m} \right\rangle_{n,i} = -\sum_{n=1}^{N_g} \omega_n \, U_i(x_n^i) \Phi(x_n^i) \, Y^m_\ell(s_n). \tag{9}$$

If this linear system is solved by means of any numerical method for non-symmetric matrices such as the generalized minimal residual (GMRes) method or coupled Jacobi/direct inversion in the iterative subspace[22] (DIIS) iterations[23], the energy can then be computed following

$$E_s = \frac{1}{2} f(\varepsilon_s) \int_\Omega \rho(x) W(x) dx \qquad \text{or, its discrete variant} \qquad E_s^N = \frac{1}{2} f(\varepsilon_s) \sum_i \sum_{\ell m} [X_i]^m_\ell [\Psi_i]^m_\ell = \frac{1}{2} f(\varepsilon_s) \langle X, \Psi \rangle,$$

where we introduced the compact notation

$$\langle A, B \rangle = \sum_i \sum_{\ell m} [A_i]^m_\ell [B_i]^m_\ell$$

and where the definition of the vector Ψ with coefficients $[\Psi_i]^m_\ell$ depends on the nature of the solute's charge, i.e. whether the COSMO is coupled with a classical, quantum mechanical or hybrid model, which will be specified in the next section.

Please also note that in order to achieve a smooth energy while varying nuclear coordinates, the weight functions ω_{ij} and U_i are replaced by smooth approximate functions in practice. To do so, let us first introduce the polynomial of degree 5 given by

$$p_\eta(t) = \eta^{-5}(1-t)^3(6t^2 + (15-12)t + 1 - \eta^2 - 15\eta + 6),$$

so that the regularized characteristic function defined by

$$\chi_\eta(t) = \begin{cases} 1 & \text{if } t \leq 1 - \eta, \\ p_\eta(t) & \text{if } 1 - \eta < t < 1, \\ 0 & \text{if } t \geq 1, \end{cases}$$

is twice continuously differentiable, in order to ensure the continuity of the forces and second-derivatives of the energy. Finally, the regularized versions of ω_{ij} and U_i are provided by the following expressions

$$\omega_\eta^{ij}(x) = d^i(x_n^i)\,\chi_\eta\left(r_1^j(x)\right) \qquad \text{resp.} \qquad U_\eta^i(x) = 1 - \sum_{j\in\mathcal{N}_i} \omega_\eta^{ij}(x) \qquad \text{with} \qquad d^i(x) = \frac{\min\left(\sum_{k\in\mathcal{N}_i}\chi_\eta\left(r_1^k(x)\right), 1\right)}{\sum_{k\in\mathcal{N}_i}\chi_\eta\left(r_1^k(x)\right)}.$$

The construction of this regularization is a delicate task and its explanation goes beyond the scope of this communication[10].

In order to compute the ddCOSMO forces or other derived quantities, analytical derivatives of the ddCOSMO energy are required. Differentiating eq .2 with respect to a generic parameter λ we get

$$\frac{dE_s}{d\lambda} = E_s^\lambda = \frac{1}{2} f(\varepsilon_s)\left(\left\langle X^\lambda, \Psi\right\rangle + \left\langle X, \Psi^\lambda\right\rangle\right), \tag{10}$$

where we denote the derivative with respect to λ with a superscript. Note that the function Ψ depends explicitly on λ, while the coefficients $[X]$ depend implicitly on λ. In order to compute the derivatives of the coefficients, we differentiate the ddCOSMO equation

$$(LX)^\lambda = L^\lambda X + LX^\lambda = g^\lambda \qquad \Rightarrow \qquad X^\lambda = L^{-1}(g^\lambda - L^\lambda X). \tag{11}$$

By substituting eq. 11 into eq. 10, we get

$$E_s^\lambda = \frac{1}{2} f(\varepsilon_s)\left(\left\langle L^{-1}(g^\lambda - L^\lambda X), \Psi\right\rangle + \left\langle X, \Psi^\lambda\right\rangle\right) = \frac{1}{2} f(\varepsilon_s)\left(\left\langle g^\lambda - L^\lambda X, (L^{-1})^\dagger \Psi\right\rangle + \left\langle X, \Psi^\lambda\right\rangle\right). \tag{12}$$

If we now introduce the ddCOSMO adjoint equations

$$L^\dagger S = \Psi, \tag{13}$$

the general expression for the analytical derivative of the ddCOSMO energy becomes

$$E_s^\lambda = \frac{1}{2} f(\varepsilon_s)\left(\left\langle g^\lambda - L^\lambda X, S\right\rangle + \left\langle X, \Psi^\lambda\right\rangle\right). \tag{14}$$

In conclusion, in order to realize the ddCOSMO algorithm, one needs to be able to compute the right-hand side (g, eq. 9) to the ddCOSMO equation (eq. 6) and to solve the linear system. Then, to compute the energy, one has to assemble the function Ψ (eq. 2). Any further computation requires to solve the ddCOSMO adjoint equation (eq. 13) and to compute the derivative of the ddCOSMO energy (eq. 14), which in turn requires the computation of the derivatives of g and Ψ, and the product between the coefficients X and the differentiated ddCOSMO matrix. In section 3, all these terms will be detailed for QM and classical solutes.

3 | COUPLING ddCOSMO TO AN EXISTING CODE

In this section, we discuss how to implement ddCOSMO in an existing code (called *the host code* in the following) by coupling such a code to the ddCOSMO module[1]. We recall that the ddCOSMO code can be obtained for free[18] under

[1] Available at `https://github.com/filippolipparini/ddPCM`

the terms of the LGPL license (version 3), which is a weak copyleft license that allows developers and companies to use, modify, redistribute and integrate the code with the only constraint that the modified code has to be made available under the terms of the same license. If a proprietary code includes ddCOSMO, only the modified ddCOSMO library has to be made available with its source code, thus protecting the intellectual property of both the authors of the library and of the company that produces the proprietary code.

First, we give an overview of the main components of a modular implementation of ddCOSMO. Then, we analyze more in the specific the quantities that the host code needs to provide to the ddCOSMO module and how the host code has to digest the output of ddCOSMO in order to proceed further. We consider two different cases. First, we analyze the calculation of the (self-consistent) energy for a solute described at a QM level of theory such as Hartree-Fock or Kohn-Sham Density Functional Theory, namely, a method that requires the solution of the self-consistent field (SCF) equations. Second, we consider the case of a classical force field energy and forces calculation, such as the one required to perform a step of molecular dynamics.

3.1 | The general picture

The most modular implementation possible would almost not require any new coding and be based on the input/output of the host code and of the ddCOSMO module, with the only coding required being limited to parsing and writing input and output files. While this is in principle possible, it would be terribly inefficient, especially when the solute is described with a QM method, or even a polarizable force field. We will therefore adopt the following philosophy. We divide the code in three sets of routines

- **Host software routines.** These routines are independent of ddCOSMO and are assumed to be already present in the host code . They perform general operations such as the evaluation of electrostatic quantities at provided grid points, the numerical quadrature for density functional theory or one and two electron integrals. No modification to such routines is needed in order to achieve the coupling.
- **ddCOSMO routines.** These routines are independent of the host code and deal exclusively with ddCOSMO-related quantities. They include setup routines, the routines needed to solve the ddCOSMO linear equations and routines that compute differentiated ddCOSMO quantities. No modification to such routines is needed in order to achieve the couplings.
- **driver routines.** These routines take care of the actual coupling by calling the appropriate host software or ddCOSMO routines. The implementation of driver routines is therefore the main task to be performed in order to achieve the coupling between ddCOSMO and the host software .

In this contribution, we only describe the functionalities that the host software routines need to provide. As these routines are strongly code-dependent we cannot provide specific details. Nevertheless, the operations to be performed are described in detail from an abstract point of view. In the following, we will discuss how to set up a driver. The ddCOSMO routines will be the subject of section 4.

3.2 | Initializing ddCOSMO

Independent of the method and level of theory to which ddCOSMO is coupled, any computation involving ddCOSMO requires some geometrical input data comprising:

TABLE 1 Suggested default values for the ddCOSMO parameters for QM and classical (MM) applications

Parameter	QM default	MM default
$\ell_{\max}$	10	6
N_g	302	110
η	0.1	0.1
`iconv`	8	6

- the number of atoms forming the molecule (`n`);
- the position of the centers of the spheres that make the cavity (`x`, `y`, `z`, arrays of size `n`);
- the radii of the spheres that make the cavity (`rvdw`, array of size `n`) (for instance, the UFF radii scaled by 1.1);

as well as a few scalar parameters

- the order $\ell_{\max}$ of the spherical harmonics expansion used in the discretization (`lmax`) ;
- the number N_g of Lebedev points to be used for the numerical integration (`ngrid`);
- the width η of the switching region relative to the radii of the spheres (`eta`);
- the solvent's dielectric constant ε (`eps`);
- the convergence threshold (`iconv`, the threshold is defined as $10^{-\text{iconv}}$);
- a flag that, if set to a nonzero value, indicates that the computation of the forces is required (`igrad`).

The variable names in our implementation of these global variables is emphasized in the parentheses. Such variables need to be initialized by the user through the host code . With these input, an internal data-structure is built that comprises

- the list $\mathcal{N}_i$ of intersecting spheres for each sphere (`nl`, see section 4.1 for further detail);
- Lebedev integration points s_n and weights ω_n (`grid` and `w`, arrays of dimension (`3,ngrid`); and (`ngrid`), respectively);
- the number and coordinates of the points that lie on a portion of the cavity which is exposed to the solvent (`ncav` and `ccav`, array of dimension (`3,ncav`));
- several quantities that are precomputed in order to speed up the computation (see section 4.1).

In a Fortran 90 implementation, these data as well as the above presented geometrical data and scalar parameters can be accessed when using our module by including `use ddcosmo` in the host code . The routine that handles the computations of this data-structure is called DDINIT and needs to be invoked before any ddCOSMO computation. It can be invoked as follows:

```
>> call ddinit(n,x,y,z,rvdw).
```

In table 1, a suggested set of default parameters is provided for both QM and classical applications. These parameters are, in our experience, safe combinations that result in a good compromise between the numerical accuracy and efficiency of ddCOSMO.

3.3 | Case 1: SCF energy with ddCOSMO

The coupling of ddCOSMO to the SCF procedure requires, at each SCF iteration, not only to compute the solvation energy and add it to the total energy, but also to add a contribution to the Fock (or Kohn-Sham) matrix. Both these contributions can be assembled right after the two-electron contribution to the Fock matrix has been computed. The main difficulty introduced by ddCOSMO in a QM computation[11] lies in the way the ddCOSMO solvation energy is computed. Contrary to the usual discretizations of COSMO, ddCOSMO does not give access to an apparent surface charge, but relies entirely on the reaction potential W, which can be computed inside each sphere locally starting from the solution to the ddCOSMO equations X.

First, one has to assemble the right-hand side for the ddCOSMO linear system, defined in eq. 9. Note that the potential Φ is defined, for a QM solute, as

$$\Phi(x_n^i) = \sum_K \frac{Z_K}{|x_K - x_n^i|} - \sum_{\mu\nu} P_{\mu\nu} \int_{\mathbb{R}^3} \frac{\chi_\mu(x)\chi_\nu(x)}{|x - x_n^i|} dx, \tag{15}$$

where we recall that $x_i^n = x_i + r_i s_n$, Z_K is the charge of the K-th nucleus and x_K its position, and $P_{\mu\nu}$ is the density matrix in the basis of the atomic orbitals $\{\chi_\mu\}$. Note that the value of the potential at the internal points is not needed, as the characteristic function of the external surface appears explicitly in the ddCOSMO right-hand side (eq. 9). The solvation energy is defined as

$$E_s = \frac{1}{2} f(\varepsilon) \int_\Omega \rho(x) W(x) dx, \tag{16}$$

where for a QM solute the density is given by

$$\rho(x) = \sum_K Z_K \delta(x - x_K) - \sum_{\mu\nu} P_{\mu\nu}\chi_\mu(x)\chi_\nu(x) = \rho^{\text{nuc}} + \rho^{\text{ele}}. \tag{17}$$

As seen in eq. 4, the reaction potential can be expressed inside a given sphere Ω_j as

$$W|_{\Omega_j}(x) \approx W_j(x) = \sum_{\ell m} \frac{4\pi}{2\ell + 1} [X_j]_\ell^m \, r_\ell^j(x) \, Y_{\ell,m}^i(x).$$

In order to take advantage of the ddCOSMO local representation of W, eq. 16 can be rewritten as a sum of contributions over the spheres. However, in order to avoid double counting contributions arising from the intersection of spheres, a weight function θ_j has to be introduced:

$$E_s = \frac{1}{2} f(\varepsilon) \sum_j \int_{\Omega_j} \rho(x) \theta_j(x) W_j(x) dx. \tag{18}$$

The integral in eq. 18 can be evaluated numerically using the same quadrature that is used for the exchange-correlation (XC) potential in density functional theory (DFT). Such a quadrature[24] is in fact performed with a set of atom-centered points and uses weights to avoid double counting. Let $\{z_{jg}, \theta_{jg}\}$ be the set of points and weights used for the DFT quadrature around the j-th atom. We can then write, for the contribution stemming from the electronic density[11]:

$$E_s = \frac{1}{2} f(\varepsilon) \sum_j \sum_g \theta_{jg} \rho^{\text{ele}}(z_{jg}) W_j(z_{jg}) = \frac{1}{2} f(\varepsilon) \sum_j \sum_g \theta_{jg} \rho(z_{jg}) \sum_{\ell m} \frac{4\pi}{2\ell + 1} r_\ell^j(z_{jg}) Y_{\ell,m}^j(z_{jg}) [X_j]_\ell^m. \tag{19}$$

If we compare eq. 19 with eq. 2, we get

$$[\Psi_j^{\text{ele}}]_\ell^m = \sum_g \theta_{jg} \rho^{\text{ele}}(z_{jg}) \frac{4\pi}{2\ell+1} r_\ell^j(z_{jg}) Y_{\ell,m}^j(z_{jg}). \tag{20}$$

The vector Ψ^{ele} can therefore be computed by using the same code that is used for XC quadrature in DFT. Note that DFT grids are usually built as the product of a radial and an angular grid. As the quantities evaluated at grid points in eq. 20 can also be clearly separated into a radial and an angular part, it is possible to precompute radial and angular intermediates and to assemble them while evaluating the integral. This makes the additional ddCOSMO cost negligible with respect to the cost of assembling the grids and evaluating the density at the grid points, two operations that are anyway needed in a DFT calculation. The nuclear contribution can be computed separately from the electronic one and added to form the total Ψ. For the nuclear charge, we get

$$[\Psi_j^{\text{nuc}}]_\ell^m = 2\sqrt{\pi} Z_j \delta_{\ell 0} \delta_{m0} \tag{21}$$

A numerical quadrature is also necessary in order to assemble the ddCOSMO contribution to the Fock matrix, that is defined as the derivative of the ddCOSMO energy with respect to the density matrix

$$F_{\mu\nu}^s = \frac{dE_s}{dP_{\mu\nu}} = \frac{1}{2} f(\varepsilon_s) \left(\left\langle \frac{\partial g}{\partial P_{\mu\nu}}, S \right\rangle + \left\langle X, \frac{\partial \Psi}{\partial P_{\mu\nu}} \right\rangle \right), \tag{22}$$

where we used eq. 14 and the fact that the ddCOSMO matrix does not depend on the density. The first contribution can be rearranged as follows:

$$\left\langle \frac{\partial g}{\partial P_{\mu\nu}}, S \right\rangle = -\sum_j \sum_n \sum_{\ell m} [S_j]_\ell^m \omega_n U_\eta^j(x_n^j) Y_\ell^m(s_n) \frac{\partial \Phi(x_n^j)}{\partial P_{\mu\nu}}. \tag{23}$$

The sum over ℓ and m in eq. 23 can be computed before the evaluation of the integrals. Introducing the new quantities

$$\xi_n^j = -U_\eta^j(x_n^j) \omega_n \sum_{\ell m} [S_j]_\ell^m Y_\ell^m(s_n), \tag{24}$$

eq. 23 becomes

$$\left\langle \frac{\partial g}{\partial P_{\mu\nu}}, S \right\rangle = -\sum_j \sum_n \xi_n^j \int_{\mathbb{R}^3} \frac{\chi_\mu(x) \chi_\nu(x)}{|x_n^j - x|} dx. \tag{25}$$

Note that ξ_i^n vanishes at points buried inside the cavity, so that it only needs to be evaluated and contracted with the integrals at the external points. The second contribution to the Fock matrix requires a second numerical integration. We get

$$\left\langle X, \frac{\partial \Psi}{\partial P_{\mu\nu}} \right\rangle = -\sum_j \sum_g \sum_{\ell m} [X_j]_\ell^m \theta_{jg} \chi_\mu(z_{jg}) \chi_\nu(z_{jg}) \frac{4\pi}{2\ell+1} r_\ell^j(z_{jg}) Y_{\ell,m}^j(z_{jg}). \tag{26}$$

Again, the sum over ℓ and m can be computed before evaluating the integral. By introducing

$$\eta_{jg} = \sum_{\ell m} \frac{4\pi}{2\ell+1} r_\ell^j(z_{jg}) Y_{\ell,m}^j(z_{jg}) [X_j]_\ell^m \tag{27}$$

we get

$$\left\langle X, \frac{\partial \Psi}{\partial P_{\mu\nu}} \right\rangle = -\sum_j \sum_g \eta_{jg} \theta_{jg} \chi_\mu(z_{jg}) \chi_\nu(z_{jg}). \tag{28}$$

We have now defined all the quantities required for a SCF/ddCOSMO calculation and can describe the structure of the driver routine. We focus on a single SCF iteration and underline the quantities that can be precomputed at the beginning. We also provide details on the specific calls to the ddCOSMO routines included in the ddCOSMO module, commenting on the interface.

1. Initial setup, as described in section 3.2.
2. Computation of the solute's potential at the cavity points. The driver needs to call the appropriate, host code specific, integral routine to compute the potential (see eq. 15) at the external points (in `ccav`).

$$\Phi_i = \sum_K \frac{Z_K}{|x_K - x_n^i|} - \sum_{\mu\nu} P_{\mu\nu} \int_{\mathbb{R}^3} \frac{\chi_\mu(x)\chi_\nu(x)}{|x - x_n^i|} dx, \tag{29}$$

   where the potential needs to be computed only at the points x_i^n that lie on the external cavity surface (i.e., the ones contained in the `ccav` array). We assume that the potential is stored in an array `phi` of size `ncav`.
3. Computation of Ψ: The driver needs to call the quadrature routines to compute the array `psi`, an array of size ((`lmax`+ 1)^2,`n`). The details of how this array is assembled are strictly dependent on the host code , but eq. 20 should be straightforward to implement given the DFT quadrature routines.
4. Solution to the ddCOSMO equation and computation of the ddCOSMO energy. This operation is easily performed by calling the COSMO routine from the ddCOSMO module. The driver needs to allocate space for the X coefficients, which are stored in an array `xcoef` of dimension ((`lmax`+1)^2, `n`). The COSMO routine can accept as an input either the ddCOSMO right-hand side (see eq. 9) or the potential at the external points, which is then processed internally. Assuming the latter case, the routine should be called as follows:

   ```
   >> call cosmo(.false.,.true.,phi, xx, psi, xcoef, esolv).
   ```

   The first argument specifies whether the ddCOSMO equation (`.false.`) or its adjoint (`.true.`) should be solved. `xx` is a dummy array that acts as a placeholder for the ddCOSMO right-hand side and is not referenced when the second argument is set to `.true.`. `esolv` is a real scalar that contains in output the ddCOSMO electrostatic solvation energy E_s.
5. Solution to the ddCOSMO adjoint equation. Again, this is achieved by calling the COSMO routine from the ddCOSMO module. The driver needs to allocate space for the S coefficients, which are stored in an array `scoef` of dimension ((`lmax`+1)^2, `n`). With respect to the previous call, the only differences are that the first logical has to be set to `.true.` and that the output has to be saved in the array scoef:

   ```
   >> call cosmo(.true.,.true.,phi, xx, psi, scoef, esolv).
   ```

6. Computation of the ddCOSMO contributions to the Fock matrix. The driver needs here to call the proper integral

and numerical quadrature routines from the host code in order to assemble the contributions in eqs. 23 and 26. For improved efficiency, the intermediates ξ and η should be precomputed and used as in eqs. 25 and 28. The ξ intermediate can be computed by invoking the DDMKXI routine from the ddCOSMO module, which is invoked with the following arguments

```
>> call ddmkxi(scoef, xi).
```

Note that an array `xi` of size `ncav` should be allocated by the user. The `xi` array can then be passed to the integrals evaluation routine. There is no routine to assemble the η intermediate in the ddCOSMO module, as it is assumed that the DFT quadrature points are not easily available in the driver. η can be evaluated on-the-fly inside the integration routine whenever a batch of integration points is processed. This can be done before the loop over the basis functions for improved efficiency.

3.4 | Case 2: Energy and forces with a MM force field

Interfacing ddCOSMO to a classical code is overall much simpler than interfacing it to a quantum mechanical code. In this section, we discuss how to compute the ddCOSMO contributions to the energy and forces when the solute is described with a classical force field. We assume that the electrostatic interactions are modeled by endowing each atom with a point charge, as is commonly done in popular force fields such as AMBER[25], CHARMM[26], OPLS[27] and others. The ddCOSMO contribution to the forces encompasses both ddCOSMO specific quantities and quantities that depend on the solute and are to be computed by a routine from the host code . Differentiating the ddCOSMO energy with respect to the nuclear coordinates of the i-th atom, we get

$$\nabla_i E_s = \langle S, \nabla_i g - (\nabla_i L) X \rangle , \tag{30}$$

as Ψ does not depends on the position of the nuclei when the solute's density of charge is a collection of point charges, as it is apparent from eq. 21. Here, ∇_i denotes the derivative with respect to the nuclear coordinate x_i. The terms stemming from the derivatives of the L matrix are ddCOSMO exclusive quantities. For the derivatives of the right-hand side, we get

$$\nabla_i [g_j]_\ell^m = -\sum_{n=1}^{N_g} \omega_n \, Y_\ell^m(s_n) \left(\nabla_i U_\eta^j(x_n^j) \Phi(x_n^j) + U_\eta^j(x_n^j) \nabla_i \Phi(x_n^j) \right) . \tag{31}$$

The first contribution is again a ddCOSMO exclusive quantity and originates from the fact that the characteristic function U_j is regularized by U_η^j, which is required to have a continuous energy as a function of the nuclear coordinates. The second term requires to compute the derivatives of the solute's potential and requires a host code routine that computes electrostatic quantities.

Let us now describe the structure of a MM/ddCOSMO driver.

1. Initial setup. This step is identical to the one described in section 3.2 as already mentioned in section 3.3 and is readily achieved by setting the ddCOSMO scalars and invoking the DDINIT routine from the ddCOSMO module.
2. Compute the solute's potential at the external cavity points. For a set of point charges q_K, the electrostatic potential

is given by

$$\Phi(x_n^i) = \sum_{K=1}^{N_a} \frac{q_K}{|x_K - x_n^i|}. \tag{32}$$

For a simple-minded implementation, a naive double-loop routine can be easily written in a few lines of code. An example of such a naive implementation is provided in the ddCOSMO module (subroutine MKRHS). However, for molecular systems as large as the ones that are usually treated with classical molecular mechanics (MM), a fast summation technique is required in order to achieve linear scaling and speed up the computation. The most obvious choice is probably to resort to the Fast Multiple Method[28] (FMM), for which many open source and highly optimized implementations exist. Nevertheless, the result of this step should be, as in section 3.3, an array `phi` of size `ncav` containing the potential at the external points.

3. Compute Ψ. This is trivially done for a classical solute, as only eq. 21 is involved. We get

$$[\Psi_j]_\ell^m = 2\sqrt{\pi} q_j \delta_{\ell 0} \delta_{m0}. \tag{33}$$

The MKRHS subroutine in the ddCOSMO module also computes Ψ.

4. Solve the ddCOSMO equations. This step is independent of the level of theory used for the solute and can be performed as described in section 3.3, by invoking

```
>> call cosmo(.false.,.true.,phi, xx, psi, xcoef, esolv).
```

Again, the ddCOSMO contribution to the solvation energy (in atomic units) is given as an output in `esolv`.

5. Solution to the ddCOSMO adjoint equation. This step is needed if the computation of the forces is required and can be achieved by calling the COSMO routine

```
>> call cosmo(.true.,.true.,phi, xx, psi, scoef, esolv).
```

6. Computation of the forces - ddCOSMO exclusive terms. These contributions stem from the derivatives of the ddCOSMO L matrix (see eq. 14) and from the derivatives of the characteristic function (see eq. 31.). The FORCES_DD subroutine from the ddCOSMO package takes care of assembling these terms. The user should allocate a temporary array for the ddCOSMO forces `fx` of dimension `(3,n)` and invoke the FORCES_DD subroutine as follows

```
>> call forces_dd(n,phi,xcoef,scoef,fx).
```

7. Computation of the forces - code-specific terms. The remaining contribution to the forces involves the derivatives of the potential. For the i-th atom, this terms contributes as follows (note that the forces are minus the derivative of the energy):

$$F_i^{\text{non dd}} = \sum_j \sum_{\ell m} [S_j]_\ell^m \sum_n w_n U_\eta^j(x_n^j) Y_\ell^m(s_n) \nabla_i \Phi(x_n^j). \tag{34}$$

Let

$$\zeta_n^j = w_n U_\eta^j(x_n^j) \sum_{\ell m} [S_j]_\ell^m Y_\ell^m(s_n). \tag{35}$$

Through simple algebra, we get

$$F_i^{\text{non dd}} = -\sum_n \zeta_n^i \sum_k \frac{q_k(x_n^i - R_k)}{|x_n^i - R_k|^3} - q_i \sum_j \sum_n \frac{\zeta_n^j(R_i - x_n^j)}{|R_i - x_n^j|^3}. \tag{36}$$

The first term can be interpreted as the product of (minus) the electric field produced by the solute's charges at the external cavity points interacting with the fictitious charge distribution ζ. The second, on the contrary, can be seen as the field produced by the distribution ζ at the atoms interacting with the MM charges. While the physical interpretation of these two terms is, of course, inessential, it makes the use of standard electrostatics machinery possible, including taking advantage of linear scaling techniques such as the FMM. A simple-minded routine (subroutine EFLD) that computes the electric field of a source at a set of target points with a double loop is included in the ddCOSMO package. Furthermore, a subroutine that, given the S coefficients produces the array ζ (subroutine DDMKZETA) is provided. Note that, due to the presence of the external characteristic function in the definition of ζ, such a quantity is only nonzero at the external cavity points, that are given in the `ccav` array. As a consequence, the two electric fields computations required to evaluate eq. 36 can be restricted to targets (sources) lying just on such points. Therefore, in order to compute this contribution, the user needs to allocate an array `zeta` of dimension ncav and fill it via

```
>> call ddmkzeta(scoef,zeta).
```

The appropriate, host code specific routine to compute electric fields are then needed in order to evaluate the "fields" in eq. 36. Finally, eq. 36 can be evaluated.

This concludes the presentation of the MM/ddCOSMO driver. The ddCOSMO package includes a prototypical implementation of a MM driver (program main, in `main.f90`), which reads the MM charges, coordinates and Van der Waals radii from a text file, initializes the ddCOSMO environment and proceeds with the steps described in this section. The implementation is extensively commented and can be used as a starting point for interfacing ddCOSMO with a classical MM code.

4 | THE ddCOSMO MODULE

In this section, a more detailed description of the implementation of ddCOSMO is provided. Besides providing a complete description of the existing open source code, this section provides guidelines for a new implementation, which can be used in alternative to the existing one for better integration in an existing code, in particular when such a code is written in a language different from Fortran 90.

There are three main operations required in order to perform a ddCOSMO computation. Such operations are mentioned as the ddCOSMO specific steps in section 3 and can be summarized as follows

1. initial setup;
2. solution to the ddCOSMO linear systems;
3. computation of the ddCOSMO specific contributions to the forces.

In the following, each of these steps is detailed.

4.1 | Initial setup of a ddCOSMO calculation

The setup of a ddCOSMO calculation requires the computation of several quantities that depend on the geometry of the solute and on the cavity (i.e., on the van der Waals radii used). Some of the quantities are precomputed for the sake of efficiency but can be also computed on-the-fly whenever needed. Given the limited amount of memory required to store such quantities, we believe that it is always worth precomputing them, as they allow for non-negligible savings in computation time.

As mentioned in section 3, the first operation that needs to be performed is to set the ddCOSMO scalars. We also set the number of spheres (`nsph`) to the number of atoms of the solute and compute `nylm=(lmax+1)^2`. The first quantity that needs to be assembled is the integration grid and weights. This requires to allocate two arrays, one for the grid (`grid`) of size (`3,ngrid`), and one for the weights (`w`) of size (`ngrid`). In our implementation, we use the publicly available routines by Laikov and van Wuellen (see `http://www.ccl.net/cca/software/SOURCES/FORTRAN/Lebedev-Laikov-Grids`). We then create two arrays, one for the centers of the sphere (`csph`) of size (`3,nsph`), that coincide with the positions of the atoms, and one for their radii (`rsph`) of size (`nsph`).

A neighbor list, defined as the list of spheres that overlap each sphere, is then computed. The neighbor list is of crucial importance, as only overlapping spheres interact or, in more specific terms, only the off-diagonal blocks of the ddCOSMO matrix that correspond to two overlapping spheres are nonzero. Therefore, the neighbor list is also used to describe the sparsity pattern of the ddCOSMO matrix. Neighbor lists can be computed using several algorithms. In our implementation, we use a simple-minded double loop algorithm. This is of course not an efficient choice, which was made for the sake of simplicity. Routines that assemble neighbor lists with a computational cost that scales linearly with respect to the number of atoms are commonly available in any MD code and open source implementations can be easily found. We store the neighbor list using two integer arrays. The first, `inl`, of dimension `nsph+1`, contains a list of integers that indicate where the list of neighbors of the i-th sphere start. The second array, `nl`, contains the actual compressed neighbor list. According to this convention, the neighbors of the i-th sphere are `nl(inl(isph))`, ..., `nl(inl(isph+1))-1`.

For the sake of convenience, we also precompute several quantities that are used repeatedly during a ddCOSMO calculation. We allocate two real arrays, `facs` and `facl`, of dimension `nylm`, that contain the normalization factors for the spherical harmonics functions (N_ℓ^M, in `facs`)

$$N_\ell^m = (-1)^m \sqrt{2} \sqrt{\frac{2\ell+1}{4\pi} \frac{(\ell-|m|)!}{(\ell+|m|)!}}$$

and the inverse diagonal of the diagonal blocks of the ddCOSMO L matrix (in all blocks are equal)

$$[L_{ii}]^{mm'}_{\ell\ell'} = \frac{2\ell+1}{4\pi} \delta_{\ell\ell'} \delta_{mm'}.$$

We then compute the set of spherical harmonics $\{Y_\ell^m(s_n)\}$, $\ell = 0, \ldots, \ell_{\max}$, $m = -\ell, \ldots, \ell$ at the grid points s_n for $n = 1, \ldots, N_g$, and save it in an array `basis` of dimension (`nylm,ngrid`). The computation of spherical harmonics at a given point on the unit sphere (performed by the subroutine `ylmbas`) is a key component of the ddCOSMO algorithm and a crucial factor in the overall efficiency of the code. The algorithm that we use is described in ref. [10] and is based on a recursion formula for the generalized Legendre polynomials (subroutine `polleg`) and on Chebyshev polynomials to evaluate the trigonometric functions (subroutine `trgev`).

Finally, we precompute two geometric quantities, that are defined at each grid point for each sphere and saved into two arrays, `ui` and `fi`, of dimension (`ngrid,nsph`). These arrays are used in order to evaluate the external and internal

characteristic functions and are defined as follows

$$f_n^i = \sum_{j \in N_i} \chi_\eta(r_1^j(x_n^i)) \qquad \text{and} \qquad u_n^i = \max(1 - f_n^i, 0) = U_\eta^i(x_n^i). \tag{37}$$

If the forces are to be computed, it is also convenient to assemble

$$z_n^i = \sum_{j \in N_i} \frac{1}{r_j} \chi'_\eta(r_1^j(x_n^i)) e^j(x_n^i), \tag{38}$$

where we introduced the unit vector $e^i(x) = \frac{x - x_i}{|x - x_i|}$ and χ'_η denotes the derivative of the one-dimensional switching function χ_η. The quantity z_n^i is stored in an array `zi` of dimension (`3,ngrid,nsph`).

4.2 | ddCOSMO linear system

Once the setup has been done, and the solute's potential and ψ have been computed, one has to solve the ddCOSMO linear system. In our implementation, this is done by the `cosmo` routine, that has been described in section 3. In this section, we describe such a routine in detail. The first task performed by the cosmo routine is to prepare the right-hand side, as in eq. 9. The solute's potential is only computed at the external cavity points, defined as the points for which u_n^i is nonzero. Therefore, the potential is first decompressed from `ncav` to (`ngrid,nsph`) and multiplied for the external characteristic function. Then, the numerical integration is carried out and the right-hand side assembled.

The ddCOSMO sparse linear system can be solved using any iterative solver capable of handling non symmetric linear systems, such as GMRES or the biconjugate gradient method (BCG). The use of a Jacobi solver[23] coupled with Pulay's direct inversion in the iterative subspace[22] (DIIS) is a viable alternative to these standard solvers. Such a strategy has proven to be optimal and is the one that we follow in the ddCOSMO module. A Fortran 90 implementation of the Jacobi/DIIS solver is provided in the ddCOSMO module. Independent of the specific solver, we assume that its implementation requires as an input the name of a subroutine that performs a matrix-vector product $y = Lx$ and possibly a routine that applies a preconditioner. Both routines are provided with a standard interface that requires as argument the length of the x and y vectors and the vectors themselves, i.e., the subroutine is defined as `subroutine matvec(n,x,y)`. For the Jacobi solver, only the off-diagonal part of the L matrix is relevant for the matrix-vector product routine, while the preconditioner applies the inverse diagonal of the L matrix to a vector and has the same interface than the matrix-vector multiplication routine.

The matrix-vector multiplication routine performs the following operation

$$[Y_i]_\ell^m = \sum_{j \in N_i} \sum_{\ell' m'} [L_{ij}]_{\ell\ell'}^{mm'} [X_j]_{\ell'}^{m'} = -\sum_{j \in N_i} \sum_{\ell' m'} \frac{4\pi}{2\ell' + 1} \sum_{n=1}^{N_g} \omega_n Y_\ell^m(s_n) \omega_\eta^{ij}(x_n^i) r_{\ell'}^j(x_n^i) Y_{\ell',m'}^j(x_n^i) [X_j]_{\ell'}^{m'}. \tag{39}$$

We implement this operation as follows. First, we compute the ℓ', m' sum, by defining the intermediate quantity, for a given pair of spheres i and $j \in N_i$,

$$v_n^{ij} = \sum_{\ell' m'} \frac{4\pi}{2\ell' + 1} r_{\ell'}^j(x_n^i) Y_{\ell',m'}^j(x_n^i) [X_j]_{\ell'}^{m'}. \tag{40}$$

We accumulate in v_n^i the contributions of all neighbors j:

$$v_n^i = -\sum_{j \in N_i} v_n^{ij}\, \omega_\eta^{ij}(x_n^i) \tag{41}$$

Finally, we take care of the sum over n:

$$[Y_i]_\ell^m = \sum_{n=1}^{N_g} \omega_n Y_\ell^m(s_n) v_n^i. \tag{42}$$

The first operation (eq. 40) is the evaluation of the potential produced by an internal multipolar distribution in the center of the j-th sphere at the points on the surface of the i-th sphere that are buried inside sphere j. Note that only these points need to be considered, as in eq. 41, v_n^{ij} is multiplied by the internal characteristic function $\omega_\eta^{ij}(x_n^i)$. The latter is computed as follows

$$\omega_\eta^{ij}(x_n^i) = d^i(x_n^i)\chi_\eta(r_1^j(x_n^i)), \quad d_n^i = \frac{\min(f_n^i, 1)}{f_n^i}, \tag{43}$$

where f_n^i is defined in eq. 37. We note that for each sphere, the matrix vector multiplication requires $O((\ell_{\max}+1)^2 N_g)$ operations, where the prefactor depends on the number of neighbors of the sphere. Therefore, the global cost of the matrix-vector multiplication is $O(N)$. Furthermore, the memory requirements for the computation are limited to the v_n^i intermediate, plus some $O((\ell_{\max}+1)^2)$ scratch space for the spherical harmonics. Thus, memory requirements are also linear in the size of the system. In our implementation, a loop over the spheres is the most external structure. For each sphere, a routine (`calcv`) is called to perform the operations in eqs. 40 and 41, then a second routine (`intrhs`) is called to perform the operations in eq. 42. The external loop can be trivially parallelized, as each sphere is treated independently.

4.3 | ddCOSMO adjoint linear system

The solution to the adjoint linear system can be obtained with the same solver used for the ddCOSMO equation. Again, the user must supply a routine that performs the matrix-vector multiplication with the transposed matrix, i.e., that computes

$$[Y_i]_\ell^m = \sum_{j \in N_i} \sum_{\ell' m'} [L_{ji}]_{\ell' \ell}^{m' m} [X_j]_{\ell'}^{m'} = -\sum_{j \in N_i} \sum_{\ell' m'} \frac{4\pi}{2\ell+1} \sum_{n=1}^{N_g} \omega_n\, Y_{\ell'}^{m'}(s_n)\, \omega_\eta^{ji}(x_n^j)\, r_\ell^i(x_n^j)\, Y_{\ell,m}^i(x_n^j)\, [X_j]_{\ell'}^{m'}. \tag{44}$$

We proceed as follows. We first carry out the ℓ', m' sum, by assembling

$$a_n^j = \sum_{\ell' m'} Y_{\ell'}^{m'}(s_n)\, [X_j]_{\ell'}^{m'}. \tag{45}$$

Then, we take care of the sum over n:

$$[u^{ij}]_\ell^m = -\sum_n \omega_n\, \omega_\eta^{ji}(x_n^j)\, r_\ell^i(x_n^j)\, Y_{\ell,m}^i(x_n^j)\, a_n^j. \tag{46}$$

Finally, we accumulate over the neighbors $j \in \mathcal{N}_i$:

$$[Y_i]_\ell^m = \sum_{j \in \mathcal{N}_i} \frac{4\pi}{2\ell+1} [u^{ij}]_\ell^m. \tag{47}$$

Again, we note that the cost of the operations required to assemble the matrix vector product is, for each sphere, $O((\ell_{\max}+1)^2 N_g)$ operations, where the prefactor depends on the number of neighbors. Therefore, the overall matrix-vector multiplication can be performed with $O(N)$ operations. As for memory, only the intermediates a_n^i and some scratch space for computing the spherical harmonics are required. The computation is performed via an external loop over the spheres. For each sphere, the intermediate a is computed and then a second routine (`adjrhs`) is called to take care of the operations in eq. 37 and 47. As for the ddCOSMO matrix-vector multiplication, the external loop is trivially parallelizable, as each sphere is treated independently

4.4 | ddCOSMO specific contributions to the forces

When computing the ddCOSMO forces, there are two different contributions that are specific to ddCOSMO. The first stems from the derivatives of the ddCOSMO matrix, the second from the derivatives of the U_i external characteristic function that appears in the right-hand side of the ddCOSMO equation. The non ddCOSMO-specific contributions have already been discussed in section 3.4. The contributions to the force on the i-th atom stemming from the gradients of the ddCOSMO matrix are

$$K^i = \sum_j \sum_{\ell m} [S_j]_\ell^m \sum_{k \in \mathcal{N}_j} \sum_{\ell' m'} (\nabla_i [L_{jk}]_{\ell\ell'}^{mm'}) [X_k]_{\ell'}^{m'}. \tag{48}$$

We require that $j = k$ or $k \in \mathcal{N}_j$ as $[L_{jk}]_{\ell\ell'}^{mm'} = 0$ otherwise. First, if $j = k$, the diagonal blocks of the L matrix do not depend on the positions of the nuclei, so no contribution to the forces arises. Second, for $k \in \mathcal{N}_j$, it is convenient to distinguish three different cases:

- $i = j$: We refer to this contribution as K_i^A;
- $i = k$: We refer to this contribution as K_i^B;
- $i \in \mathcal{N}_j$ but $i \neq k$: This last contribution is referred to as K_i^C.

The derivation is very cumbersome and can be found in ref. [10]. We report here the final expressions. For the first contribution, we get

$$\begin{aligned}\mathbf{K}_i^A = -\sum_{\ell m} [S_i]_\ell^m \sum_{k \in \mathcal{N}_i} \sum_{\ell' m'} \frac{4\pi}{2\ell'+1} \sum_{n=1}^{N_g} \omega_n Y_\ell^m(s_n) \Bigg[\frac{1}{r_k} \omega_\eta^{ik}(x_n^i) \delta_{l'\geq 1} r_{\ell'-1}^k(x_n^i) \left(\ell' Y_{\ell',m'}^k(x_n^i) e^k(x_n^i) + \nabla_i Y_{\ell',m'}^k(x_n^i) \right) \\ + \frac{1}{r_k} r_{\ell'}^k(x_n^i) Y_{\ell',m'}^k(x_n^i) d^i(x_n^i) \chi_\eta'(r_1^k(x_n^i)) e^k(x_n^i) - \delta_{f_n^i > 1} r_{\ell'}^k(x_n^i) Y_{\ell',m'}^k(x_n^i) d^i(x_n^i) \omega_\eta^{ik}(x_n^i) Z_n^i \Bigg] [X_k]_{\ell'}^{m'},\end{aligned} \tag{49}$$

where we introduced the notation

$$\delta_{x>y} = \begin{cases} 1 & x > y, \\ 0 & x \leq y. \end{cases}$$

The second and third contributions read

$$K_i^B = \sum_{j\in N_i}\sum_{\ell m}[S_j]_\ell^m \sum_{\ell' m'}\frac{4\pi}{2\ell'+1}\sum_{n=1}^{N_g}\omega_n Y_\ell^m(s_n)\frac{1}{r_i}\Big[\omega_\eta^{ji}(x_n^j)\delta_{\ell'\geq 1} r_{\ell'-1}^i(x_n^j)\Big(\ell' Y_{\ell' m'}^i(x_n^j)e^i(x_n^j) + \nabla_i Y_{\ell',m'}^i(x_n^j)\Big) + \Big(1-\delta_{f_n^j>1}\omega_\eta^{ji}(x_n^j)\Big)d^j(x_n^j)\chi_\eta'(r_1^i(x_n^j))r_{\ell'}^i(x_n^j)Y_{\ell',m'}^i(x_n^j)e^i(x_n^j)\Big][X_i]_{\ell'}^{m'}, \tag{50}$$

$$K_i^C = -\sum_{j\in N_i}\sum_{\ell m}[S_j]_{\ell m}\sum_{k\in N_j\setminus\{i\}}\sum_{\ell' m'}\frac{4\pi}{2\ell'+1}\sum_{n=1}^{N_g}\omega_n Y_\ell^m(s_n)\frac{1}{r_i}\delta_{f_n^j>1}d^j(x_n^j)\omega_\eta^{jk}(x_n^j)r_{\ell'}^k(x_n^j)Y_{\ell',m'}^k(x_n^j)\chi_\eta'(r_1^k(x_n^j))e^k(x_n^j)[X_k]_{\ell'}^{m'}. \tag{51}$$

We implement these contributions as follows. All three terms share the fact that the sum over ℓ, m can be precomputed. As a consequence, we assemble before the computation of the forces a vector

$$a_n^i = \sum_{\ell m}[S_j]_\ell^m Y_\ell^m(s_n).$$

We assemble separately the contribution in eq. 49 and the ones from 50 and eq. 51, the latter being treated together. Let us start with eq. 49. We take care of the ℓ', m' sums first, by assembling, for each grid point and each neighbor k of the i-th sphere,

$$\alpha_n^{ik} = \sum_{\ell' m'}\delta_{l'\geq 1}\frac{4\pi}{2\ell'+1}r_{\ell'-1}^k(x_n^i)\left(\ell' Y_{\ell' m'}^k(x_n^i)e^k(x_n^i) + \nabla_i Y_{\ell',m'}^k(x_n^i)\right)[X_k]_{\ell'}^{m'}, \tag{52}$$

$$\beta_n^{ik} = \sum_{\ell' m'}\frac{4\pi}{2\ell'+1}r_{\ell'}^k(x_n^i)Y_{\ell',m'}^k(x_n^i)[X_k]_{\ell'}^{m'}. \tag{53}$$

All contributions in eq. 52 and 53 contain quantities that have already been used for the computation of a matrix-vector product for the ddCOSMO linear system, with the exception of the spherical harmonics gradients. We compute all the $(\ell_{\max}+1)^2$ spherical harmonics and their gradients at the same time, in order to reuse intermediates as much as possible, with an algorithm described in ref. [10] and implemented in the `dbasis` routine. We emphasize again the key importance of this step for the overall efficiency of the computation, as it has a tremendous impact on the performance of the implementation. Note that the computation of β corresponds to computing the potential of the multipolar distribution $[X_k]$ at the point x_n^i. Such an operation is also required for the ddCOSMO matrix-vector products. Once the intermediate quantities α_n^{ik} (a vector of dimension 3) and β_n^{ik} (a scalar) have been computed, we accumulate over the neighbors k, introducing the vector (dimension 3) $[v_A]_n^i$ defined as follows:

$$[v_A]_n^i = \sum_{k\in N_i}\frac{1}{r_k}\omega_\eta^{ik}(x_n^i)[\alpha_A]_n^{ik} + d^i(x_n^i)\left(\frac{1}{r_k}\chi_\eta'(r_1^k(x_n^i))e^k(x_n^i) - \delta_{f_n^i>1}\omega_\eta^{ik}(x_n^i)Z_n^i\right)[\beta_A]_n^{ik}. \tag{54}$$

It is particularly important to stress that the contributions in eq. 54 are very often zero. In particular, no contribution arises for points on sphere i that are not buried in sphere k. Furthermore, the derivative of the $\chi_\eta(t)$ switching functions are nonzero only if $1-\eta < t < 1$. Therefore, many points in the n loop can be skipped, either by checking whether $0 < u_n^i = U_\eta^i(x_n^i) < 1$, which is equivalent to requiring that the point is buried, and by checking whether $r_1^k(x_n^i)$ is between

$1-\eta$ and 1. Finally, we accumulate over n, skipping the points for which we know *a priori* that there is no contribution:

$$K_i^A = -\sum_{n=1}^{N_g} \omega_n a_n^i [v_a]_n^i. \tag{55}$$

This contribution is handled by the `fdoka` subroutine in the ddCOSMO module. The cost of this contribution is, for each atom, $O((\ell_{\max}+1)^2 N_g)$, where the prefactor depends on the number of neighbors.

The second and third contributions are handled as follows. First, we assemble the intermediates for all neighbors j of the i-th sphere,

$$\alpha_n^{ji} = \sum_{\ell' m'} \delta_{l'\geq 1} \frac{4\pi}{2\ell'+1} r_{\ell'-1}^i(x_n^j) \left(\ell' Y_{\ell',m'}^i(x_n^j) e^i(x_n^j) + \nabla_i Y_{\ell',m'}^i(x_n^j) \right) [X_i]_{\ell'}^{m'}, \tag{56}$$

$$\beta_n^{ji} = \sum_{\ell' m'} \frac{4\pi}{2\ell'+1} r_{\ell'}^i(x_n^j) Y_{\ell',m'}^i(x_n^j) [X_i]_{\ell'}^{m'}. \tag{57}$$

While we are looping over the grid points and the neighbors j to i, we compute t_n^j. If such a quantity is larger than one, we loop over the neighbors k to j that are different from i and compute $r_1^k(x_n^j)$. If $1-\eta < r_1^k(x_n^j) < 1$, there is a contribution from eq. 51. In order to take care of the latter, we assemble the intermediate

$$\beta_n^{jk} = \sum_{\ell' m'} \frac{4\pi}{2\ell'+1} r_{\ell'}^k(x_n^j) Y_{\ell',m'}^k(x_n^j) [X_k]_{\ell'}^{m'} \tag{58}$$

and we accumulate

$$[v_c]_n^{ij} = -\delta_{t_n^j>1} \sum_{k\in N_j\setminus\{i\}} \frac{1}{r_i} d^j(x_n^j) \omega_\eta^{jk}(x_n^j) \chi_\eta'(r_1^k(x_n^j)) e^k(x_n^j) \beta_n^{jk}. \tag{59}$$

Then, we accumulate over the neighbors j

$$[v_B]_n^i = \sum_{j\in N_i} \left[\frac{1}{r_i} \omega_\eta^{ji}(x_n^j) [\alpha_n^{ji}] + \left(1-\delta_{t_n^j>1}\right) \omega_\eta^{ji}(x_n^j) d^j(x_n^j) \chi_\eta'(r_1^i(x_n^j)) e^i(x_n^j) \beta_n^{ji} + [v_c]_n^{ji} \right] a_n^j. \tag{60}$$

Finally, we accumulate over n and get

$$K_i^B + K_i^C = \sum_n \omega_n [v_B]_n^i. \tag{61}$$

Again, it is important to note that many terms vanish and can be skipped using the same arguments used for the first one. These contributions are computed by the `fdokb` subroutine in the ddCOSMO module. The cost of this contribution is, for each atom, $O((\ell_{\max}+1)^2 N_g)$ operations, where the prefactor depends on the number of neighbors. This second computation is overall slightly more expensive than the first one, although it maintains the same scaling. This concludes the computation of the contribution arising from the derivatives of the ddCOSMO matrix.

The last contribution to the ddCOSMO forces is

$$G_i = -\sum_{\ell m} \sum_j \sum_{n=1}^{N_g} \omega_n Y_\ell^m(s_n) (\nabla_i U_\eta^j(x_n^j)) \Phi(x_n^j) [S_j]_\ell^m. \tag{62}$$

Using the same a_n^j intermediate used for the other contributions to the forces, we get

$$G_i = -\sum_j \sum_{n=1}^{N_g} \omega_n \left(\nabla_i U_\eta^j(x_n^j)\right)\Phi(x_n^j)a_n^j.$$

The gradient of U_n^j is nonzero only if $j = i$ or $j \in \mathcal{N}_i$. In the former case, we get

$$[v_D]_n^{ii} = -\Phi_n^i Z_n^i. \tag{63}$$

In the latter case, we get

$$[v_D]_n^{ij} = \frac{1}{r_i}\Phi_n^j \chi_\eta'(r_1^i(x_n^j))e^i(x_n^j). \tag{64}$$

We can now accumulate

$$[v_D]_n^i = [v_D]_n^{ii} a_n^i + \sum_{j\in\mathcal{N}_i} [v_D]_n^{ij} a_n^j. \tag{65}$$

We finally accumulate over n and get

$$G_i = \sum_n \omega_n [v_D]_n^i. \tag{66}$$

The computation of this contribution is carried out by the `fdoga` subroutine. Its computational cost is $O(N_g)$, with a proportionality factor that depends on the number of neighbors.

We conclude this section by pointing out that the computation is performed independently for each atom (sphere), so that it can be trivially parallelized. Furthermore, the overall computation scales linearly with the number of atoms.

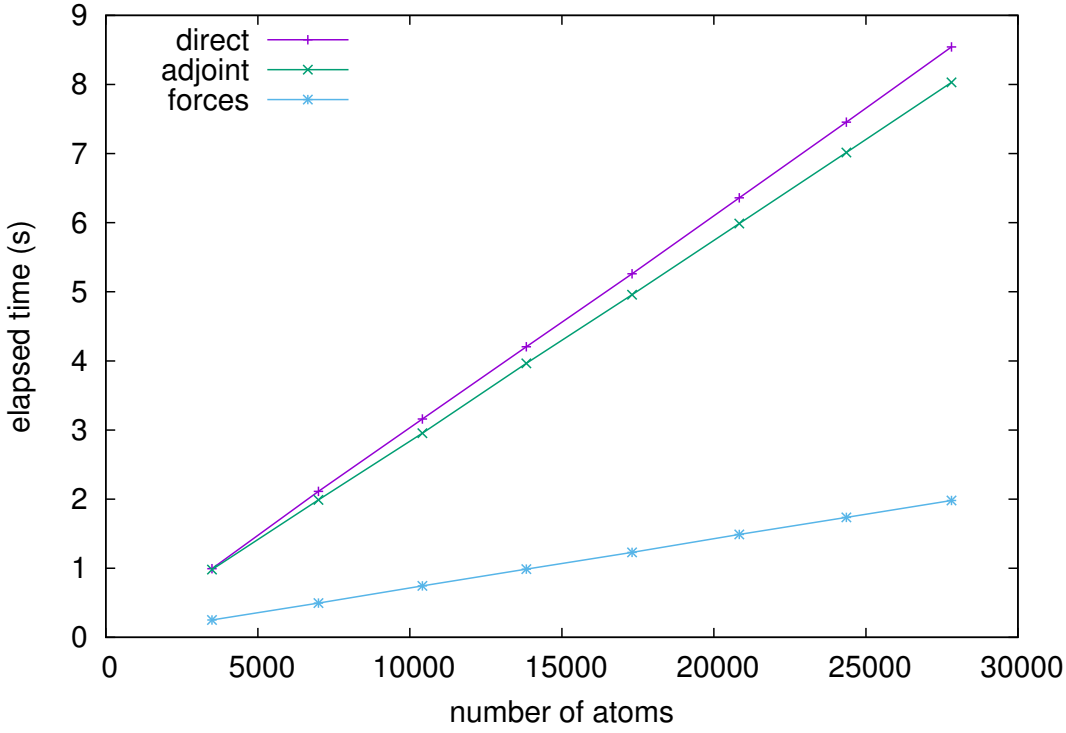
5 | CONCLUSIONS AND PERSPECTIVES

In this Software news report, we have briefly recapitulated the derivation and described in detail the implementation of ddCOSMO, a new paradigm for the solution of the Conductor-like Screening Model equations. We have provided a complete description of our ddCOSMO implementation and commented extensively on efficiency and memory issues. Furthermore, by making explicit reference to our implementation we have provided an extensive documentation and guidelines to interface it with an existing code, either for quantum chemistry or classical molecular dynamics simulation.

The modular ddCOSMO code is written in Fortran 90. Interfacing it with existing Fortran codes is thus straightforward and can be done by following the guidelines given in section 3. The same guidelines can be used, with some additional technical work, to interface the ddCOSMO module with codes written in a different language such as C, C++, python and others. The detailed description of the implementation provided in section 4, together with the existing Fortran code that can be used as a reference, are meant to provide a thorough guide in support of a new implementation, possibly in a different language.

The implementation of ddCOSMO is not a straightforward task. However, the new domain-decomposition paradigm offers several advantages with respect to other implementations. First, the discretization is systematically improvable and controlled by a very limited number of parameters, which makes it easy to have a detailed control from a numerical

FIGURE 1 Timings (in seconds) for the iterative solution of the ddCOSMO equation (direct), adjoint equation (adjoint) and for the computation of the ddCOSMO contribution to the forces (forces). Computations performed on a desktop computer equipped with an Intel i7-Q7700K quad-core processor running at 4.20GHz and 16GB of RAM. The following discretization parameters have been used: $\ell_{\max} = 6$, $N_{\text{grid}} = 110$, $\eta = 0.2$. We used a convergence threshold of 10^{-7} for the iterative solver.



point of view. Second, thanks to the regularization adopted, the continuity of the energy and forces with respect to the position of the nuclei are guaranteed, which allows for energy-conserving molecular dynamics simulations and smooth geometry optimizations. Third, it is an intrinsically linear-scaling algorithm and does not require the use of fast summation techniques, that complicate the implementation and introduce a potential source of numerical errors. Finally, and most importantly, it is overall much faster - up to three orders of magnitude - with respect to any other existing implementation. While an extensive benchmark of ddCOSMO is beyond the aims of this software review and can be found elsewhere, we report in figure 1 the timings to solve the ddCOSMO equation and adjoint equation and to compute the ddCOSMO contribution to the forces for larger and larger systems. For this example, we choose the first eight chains of the GFP-like fluorescent protein DRONPA (PDB reference 2GX2), which is composed of three tetramers, each in turn made of four chains. The calculations have been performed on a desktop computer equipped with an Intel Core i7-Q7700K quad-core processor running at 4.20GHz and equipped with 16GB of RAM. We would like to stress that, to the best of our knowledge, COSMO computations on systems as large as the ones here reported have never been attempted with any other implementation, as they would be far too expensive and memory demanding. With ddCOSMO, such computations can be routinely performed on a standard desktop computer.

In conclusion, ddCOSMO is a new, powerful algorithm that extends the applicability of polarizable continuum solva-

tion methods to large and very large systems. Currently, it has been implemented for molecular dynamics simulation with standard and polarizable force fields [13] and for QM and QM/MM calculations [7, 11] of the energy, structure, electric response properties and excitation energies. A distributed memory parallel implementation of ddCOSMO is available in the highly scalable Tinker HP molecular dynamics software [29].

The domain decomposition paradigm has recently been extended [19, 20] to the Integral Equation Formalism - Polarizable Continuum Model (IEF-PCM) [30] and to the solution of the linearized Poisson-Boltzmann equation [31]. Extensions of the domain decomposition paradigm to the Solvent Excluded Surface (SES) type of molecular surface [32, 33] has also been developed [34].

The extension of ddCOSMO and its generalizations to other molecular properties is being actively investigated in our groups. Better and more efficient parallel implementations are also being investigated, including a CPU implementation based on MPI and a GPU implementation.

ACKNOWLEDGEMENTS

Benjamin Stamm acknowledges the funding from the German Academic Exchange Service (DAAD) from funds of the "Bundesministeriums für Bildung und Forschung" (BMBF) for the project Aa-Par-T (Project-ID 57317909). Yvon Maday acknowledges the funding from the PICS-CNRS and the PHC PROCOPE 2017 (Project N° 37855ZK). This work was made possible thanks to the French state funds managed by the CalSimLab LABEX and the ANR within the Investissements d'Avenir program (reference ANR-11-LABX-0037-01).

REFERENCES

[1] Tomasi J, Mennucci B, Cammi R. Quantum Mechanical Continuum Solvation Models. Chem Rev 2005;105(8):2999–3093.

[2] Mennucci B. Continuum Solvation Models: What Else Can We Learn from Them? J Phys Chem Lett 2010;1(10):1666–1674. And references therein.

[3] Cramer CJ, Truhlar DG. Implicit Solvation Models: Equilibria, Structure, Spectra and Dynamics. Chem Rev 1999;99(8):2161–2200.

[4] Mennucci B. Polarizable continuum model. WIREs Comput Mol Sci 2012;2(3):386–404.

[5] Lipparini F, Mennucci B. Perspective: Polarizable continuum models for quantum-mechanical descriptions. J Chem Phys 2016;144(16):160901.

[6] Klamt A, Schüürmann G. COSMO: a new approach to dielectric screening in solvents with explicit expressions for the screening energy and its gradient. J Chem Soc, Perkin Trans 2 1993;p. 799–805.

[7] Lipparini F, Lagardère L, Scalmani G, Stamm B, Cancès E, Maday Y, et al. Quantum calculations in solution for large to very large molecules: A new linear scaling QM/continuum approach. The journal of physical chemistry letters 2014;5(6):953–958.

[8] Still WC, Tempczyk A, Hawley RC, Hendrickson T. Semianalytical treatment of solvation for molecular mechanics and dynamics. J Am Chem Soc 1990;112(16):6127–6129.

[9] Cancès E, Maday Y, Stamm B. Domain decomposition for implicit solvation models. The Journal of Chemical Physics 2013;139(5):054111.

[10] Lipparini F, Stamm B, Cancès E, Maday Y, Mennucci B. Fast domain decomposition algorithm for continuum solvation models: Energy and first derivatives. Journal of Chemical Theory and Computation 2013;9(8):3637–3648.

[11] Lipparini F, Scalmani G, Lagardère L, Stamm B, Cancès E, Maday Y, et al. Quantum, classical, and hybrid QM/MM calculations in solution: General implementation of the ddCOSMO linear scaling strategy. The Journal of Chemical Physics 2014;141(18):184108.

[12] Scalmani G, Barone V, Kudin K, Pomelli C, Scuseria G, Frisch M. Achieving linear-scaling computational cost for the polarizable continuum model of solvation. Theor Chem Acc 2004 Mar;111(2-6):90–100.

[13] Lipparini F, Lagardère L, Raynaud C, Stamm B, Cancès E, Mennucci B, et al. Polarizable molecular dynamics in a polarizable continuum solvent. Journal of Chemical Theory and Computation 2015;11(2):623–634.

[14] Cammi R, Tomasi J. Remarks on the use of the apparent surface-charges (ASC) methods in solvation problems - Iterative versus matrix-inversion procedures and the renormalization of the apparent charges. J Comput Chem 1995;16(12):1449–1458.

[15] York D, Karplus M. A smooth solvation potential based on the conductor-like screening model. J Phys Chem A 1999;103(50):11060–11079.

[16] Scalmani G, Frisch MJ. Continuous surface charge polarizable continuum models of solvation. I. General formalism. J Chem Phys 2010;132(11):114110.

[17] Lange AW, Herbert JM. A smooth, nonsingular, and faithful discretization scheme for polarizable continuum models: The switching/Gaussian approach. J Chem Phys 2010;133(24):244111.

[18] Lipparini F, Stamm B, Cancès E, Maday Y, Gatto p, Piquemal JP, et al., A fast domain decomposition based implementation of the COSMO solvation model;. `https://github.com/filippolipparini/ddPCM`, doi: 10.5281/zenodo.1226641.

[19] Stamm B, Cancès E, Lipparini F, Maday Y. A new discretization for the polarizable continuum model within the domain decomposition paradigm. The Journal of Chemical Physics 2016;144(5):054101.

[20] Gatto P, Lipparini F, Stamm B. Computation of Forces arising from the Polarizable Continuum Model within the Domain-Decomposition Paradigm. The Journal of Chemical Physics 2017;147(22):224108.

[21] Lebedev VI, Laikov DN. A quadrature formula for te sphere of the 131st algebraic order of accuracy. Dokl Math 1999;59:477–481.

[22] Pulay P. Convergence acceleration of iterative sequences. the case of scf iteration. Chem Phys Lett 1980;73(2):393–398.

[23] Lipparini F, Lagardère L, Stamm B, Cancès E, Schnieders M, Ren P, et al. Scalable Evaluation of Polarization Energy and Associated Forces in Polarizable Molecular Dynamics: I. Toward Massively Parallel Direct Space Computations. J Chem Theory Comput 2014;10(4):1638–1651.

[24] Becke AD. A multicenter numerical integration scheme for polyatomic molecules. J Chem Phys 1988;88(4):2547–2553.

[25] Cornell W, Cieplak P, Bayly C, Gould I, Merz K, Ferguson D, et al. A 2nd Generation Force-Field for the Simulation of Proteins, Nucleic-Acids, and Organic-Molecules. J Am Chem Soc 1995;117(19):5179–5197.

[26] Brooks BR, Bruccoleri RE, Olafson BD, States DJ, Swaminathan S, Karplus M. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. J Comput Chem 1983;4(2):187–217.

[27] Jorgensen WL, Maxwell DS, Tirado-Rives J. Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids. J Am Chem Soc 1996;118(45):11225–11236.

[28] Greengard L, Rokhlin V. A fast algorithm for particle simulations. J Comput Phys 1987;73(2):325–348.

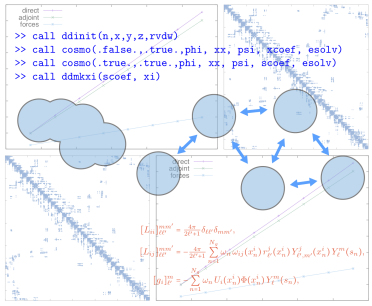
[29] Lagardere L, Jolly LH, Lipparini F, Aviat F, Stamm B, Jing ZF, et al. Tinker-HP: a massively parallel molecular dynamics package for multiscale simulations of large complex systems with advanced point dipole polarizable force fields. Chem Sci 2018;9:956–972.

[30] Cancès E, Mennucci B, Tomasi J. A new integral equation formalism for the polarizable continuum model: Theoretical background and applications to isotropic and anisotropic dielectrics. J Chem Phys 1997;107(8):3032–3041.

[31] Quan C, Stamm B, Maday Y. A Domain Decomposition Method for the Poisson-Boltzmann Solvation Models; 2018.

[32] Quan C, Stamm B. Mathematical analysis and calculation of molecular surfaces. Journal of Computational Physics 2016;322:760–782.

[33] Quan C, Stamm B. Meshing molecular surfaces based on analytical implicit representation. Journal of Molecular Graphics and Modelling 2017;71:200–210.

[34] Quan C, Stamm B, Maday Y. Polarizable Continuum Model based on the Solvent Excluded Surface. Math Mod Meth Appl S (M3AS), in press 2018;.

GRAPHICAL ABSTRACT