PARALLEL PROCESSING OF ANALYTICAL POISSON-BOLTZMANN USING HIGHER ORDER FEM

ABSTRACT
We focus on the efficient parallel processing of meshes from biomolecular data so that they can be subsequently used for FEM simulation. All mesh processing about refinements and coherent indexations are applied in parallel. The simplices of the mesh are needed for the application in FEM having higher polynomial degrees. For biomolecular data, the only inputs are the atom coordinates, the van der Waals radii and the probe radius. Our principal goal is to obtain data which are well balanced among the different processors. To corroborate the parallel mesh processing, we show some application to FEM simulation. For that, we consider the parallel FEM of the linearized Poisson-Boltzmann problem. We compare the FEM numerical results with known theoretical prediction to validate the accuracy of the parallel implementation.

KEY WORDS
Mesh, Poisson-Boltzmann, Parallel, Biomolecular.

1 Introduction

Parallel computing using multiply distributed systems have become [1, 2] more and more important since the sizes of many computational problems have grown significantly in different disciplines. Our interest here is to process tetrahedral meshes in a parallel manner so that they can be subsequently applied to distributed FEM simulations for biochemical simulation. Only the coarse mesh is managed by the master processor. All subsequent mesh processings are performed in a distributed manner. Each processor applies its mesh refinements in parallel. Since higher simplices are required for higher order FEM simulation, their assembly are equally done in a parallel manner. The domain of simulation $\mathbf{\Omega}$ is the union $\mathbf{\Omega^u} \cup \mathbf{\Omega^v}$ of the solute $\mathbf{\Omega^u}$ and the solvent $\mathbf{\Omega^v}$ that are separated by a closed surface $\mathbf{\Gamma}$ which is in applications a molecular surface (Fig. 1(b)). The entire mesh $\mathcal{M}$ is the union of two tetrahedral discretizations $\mathcal{M}^u$ and $\mathcal{M}^v$ belonging respectively to the internal domain $\mathbf{\Omega^u}$ and to the external domain $\mathbf{\Omega^v}$ with respect to the closed surface $\mathbf{\Gamma}$. In this paper, the meshes $\mathcal{M}^u$ and $\mathcal{M}^v$ are matching at the interface molecular surface. The external mesh $\mathcal{M}^v$ is bounded by a certain box.

Before presenting our results, related works are in order. The generation of a mesh on CAD data were treated in a large number of papers (see [3] and the references there). Holst [4] is one of the most prominent specialists of Poisson-Boltzmann using FEM. Most methods based exclusively on BEM (Boundary Element Methods) which treat the Poisson-Boltzmann equation (PBE) consider only the linear PBE because it is difficult to use fundamental solutions for nonlinear PBE. Although BEM can be made very efficient [5, 6], it must be combined with other methods in case of nonlinearity. The Finite Difference Method (FDM) is also widely used in PBE. The main reason does not seem to be attributed to its numerical efficiency but rather to code availability and to reference or comparison purpose. Meshfree approaches [7] are also efficient for the solving of partial differential equations but they are difficult to implement.

2 Discontinuous Coefficients in Linearized Poisson-Boltzmann

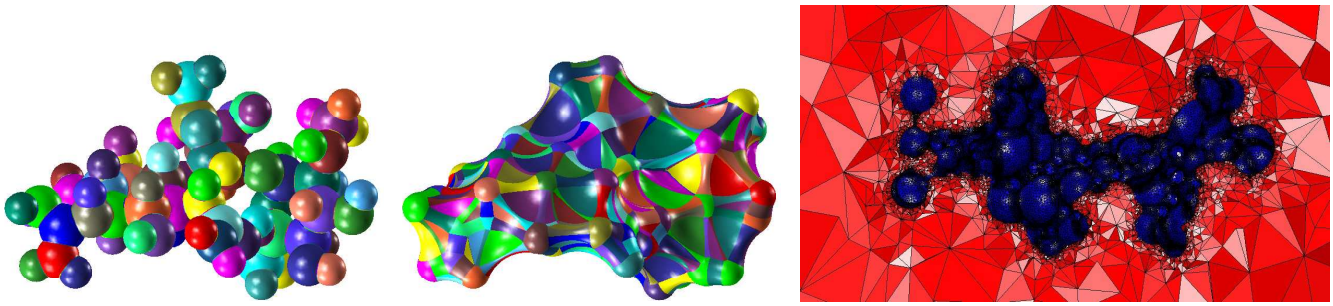
We consider the interaction between solute and solvent media. The solute $\mathbf{\Omega^u}$ (*resp.* solvent $\mathbf{\Omega^v}$) is the region located inside (*resp.* outside) a closed surface $\mathbf{\Gamma}$. In the sequel, the whole solute-solvent domain is denoted by $\mathbf{\Omega} = \mathbf{\Omega^u} \cup \mathbf{\Omega^v}$. We consider the linearized Poisson-Boltzmann Equation which is frequently met in different areas including: plasma physics, ionic solution, charged macroparticles, FET and MOSFET. Its expression is

$$
\begin{aligned}
&-\nabla\cdot\big(\varepsilon(\mathbf{x})\nabla u(\mathbf{x})\big)+\overline{\kappa}^2(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x}) \qquad \forall \mathbf{x}\in\mathbf{\Omega}.\\
&\big\{u\big\}_{\mathbf{\Gamma}}(\mathbf{x}) = 0 \qquad \forall\,\mathbf{x}\in\mathbf{\Gamma}\\
&u(\mathbf{x}) = g(\mathbf{x}) \qquad \forall\,\mathbf{x}\in\partial\mathbf{\Omega^v}\setminus\mathbf{\Gamma}
\end{aligned}
$$

in which we use the jump function

$$\big\{u\big\}_{\mathbf{\Gamma}}(\mathbf{x}) := \lim_{t\to 0^+} u\big(\mathbf{x}+t\mathbf{n}(\mathbf{x})\big) - \lim_{t\to 0^-} u\big(\mathbf{x}+t\mathbf{n}(\mathbf{x})\big)$$

where $\mathbf{n}(\mathbf{x})$ designates the normal vector at $\mathbf{x}\in\mathbf{\Gamma}$. That is, the solution is continuous everywhere. In this article, we let the right hand side function f be general but for chemical simulation, it is $f(\mathbf{x}) = (4\pi e_C^2/k_BT)\sum_{i=1}^N z_i\delta(\mathbf{x}-\mathbf{x}_i)$ in which the electric charge at $\mathbf{x}_i$ is $(4\pi e_C^2/k_BT)z_i$, the absolute temperature is T and the Boltzmann constant is k_B while e_C is the elementary charge carried by a single proton. The unknown function is the dimensionless electrostatic potential u is related to the electrostatic potential Φ by $u(\mathbf{x}) = e_C\Phi(\mathbf{x})/k_BT$. In a real chemical simulation, we set $g \equiv 0$ but for the sake of numerical comparison with generated exact solutions, we will consider a general boundary function g which might be nonzero. In general, the coefficients $\varepsilon(\mathbf{x})$ and $\kappa(\mathbf{x})$ are space-dependent functions related to the dielectric value and the modified Debye-Hückle parameter. But, in most applied cases, $\varepsilon(\mathbf{x})$ and



(a) (b) (c)

Figure 1. (a) Molecular representation, (b) SES surface $\mathbf{\Gamma}$, (c) Tetrahedral mesh in the vicinity of a molecule

$\kappa(\mathbf{x})$ are supposed to be domain-wise constants ε_u, ε_v, κ_u, κ_v between $\mathbf{\Omega^u}$ and $\mathbf{\Omega^v}$.

3 Distributed Higher Order FEM

3.1 Brief Survey about Molecular Data

Although this article focuses on the FEM mesh processing, we summarize below anyhow the acquirement of molecular data. The entire details can be found in [5, 8, 9]. We review first the problem of SES representation [10] of a cavity $\mathbf{\Gamma}$ which is the surface separating the solvent from the solute as illustrated in Fig.1(b). We focus on the domain surrounding a molecular surface acquired from digitized atomic positions $\mathbf{x}_i$ and their corresponding Van der Waals radii as displayed in Fig.1(a). The cavity $\mathbf{\Gamma}$ comes from the boundary of a molecule where each constituting atom is represented as an imaginary sphere $B(\mathbf{x}_k, r_k)$ whose center $\mathbf{x}_k$ corresponds to the atom coordinates and whose radius r_k to a multiple of the van der Waals radius. Such information are usually obtained from PDB files (Protein Data Bank). That is, the molecule is represented as the union of spheres $\bigcup_{k=1}^{N} B(\mathbf{x}_k, r_k)$. Apart from the initial molecule, we need also a probe atom. Additional requirements are needed to avoid surface self-intersections [5]. The SES model (Surface Excluded Surface) is the surface $\mathbf{\Gamma}$ traced by the probe atom when it is rolled over the whole surface $\mathcal{S} := \partial\Big[\bigcup_{k=1}^{N} B(\mathbf{x}_k, r_k)\Big]$ as illustrated in Fig. 1(b). Next, we summarize the main stages for obtaining the tetrahedralization of digitized biomolecular data. The Laguerre decomposition is a splitting of the space into non-overlapping convex polyhedra called Laguerre cells which could be bounded or unbounded. That is achieved by using the power distance which is a weighted distance function based on the position of the atoms and the Van der Waals radii. To obtain the Laguerre decomposition, one considers a certain uplifting function in order to enrich $\mathbf{x}_i = (x_i, y_i, z_i)$ by a fourth coordinate into $\widetilde{\mathbf{x}}_i = (x_i, y_i, z_i, t_i)$. One generates the convex hull $\mathcal{H}$ of the set of four dimensional points $\{\widetilde{\mathbf{x}}_i\}$. The projection of the lower face of $\mathcal{H}$ on the space $\mathbb{R}^3$ generates a weighted Delaunay tetrahedral decomposition whose dual is the Laguerre decomposition. The next step is to generate the trimmed molecular surfaces as illustrated in Fig. 1(b). That is achieved by deducing NURBS representation from the former Laguerre decomposition. Trimmed surfaces are obtained from atomic positions and spherical surfaces where the parametrization method uses stereographic projection. Afterwards, one generates a triangular surface mesh for the interface surface $\mathbf{\Gamma}$. The surface mesh generation uses Riemannian based Delaunay and NURBS mappings. The main difference with the usual Delaunay is that one employs generalized distances and angles by using Riemannian metric as detailed in [11]. The generation of the boundary B_{bound} of the whole tetrahedral decomposition is achieved by enlarging the bounding box of the whole molecular surface. One generates afterwards a coarse triangular decomposition of B_{bound}. We need initially two tetrahedral meshes $\mathcal{M}^u$ and $\mathcal{M}^v$ for the solute and the solvent respectively. The required coarse mesh $\mathcal{M}$ for the FEM simulation is the merging of $\mathcal{M}^u$ and $\mathcal{M}^v$. Those two meshes should be subject to boundary constraints. That is, at the interface surface mesh of $\mathbf{\Gamma}$, every edge (*resp.* node, *resp.* triangle) of $\mathbf{\Gamma}$ has to be an edge (*resp.* node, *resp.* triangle) of $\mathcal{M}^v$ and $\mathcal{M}^u$. That is achieved by employing the algorithm of CDT (Constrained Delaunay Tetrahedralization). In many cases, the results of a CDT contain some *slivers* which are some flaws such as very thin or flat tetrahedra. This fact is not really to be blamed on any implementation of the CDT because it is known that the CDT algorithm itself is very sensible to numerical floating operations. As a consequence, the only possible remedy is to remove those slivers.

3.2 Higher Order Data Distribution

We would like first to describe the correlation between the FEM We would like first to describe the correlation between the FEM polynomial degree and the geometric information: nodes (0-simplex), edges (1-simplex), triangu-

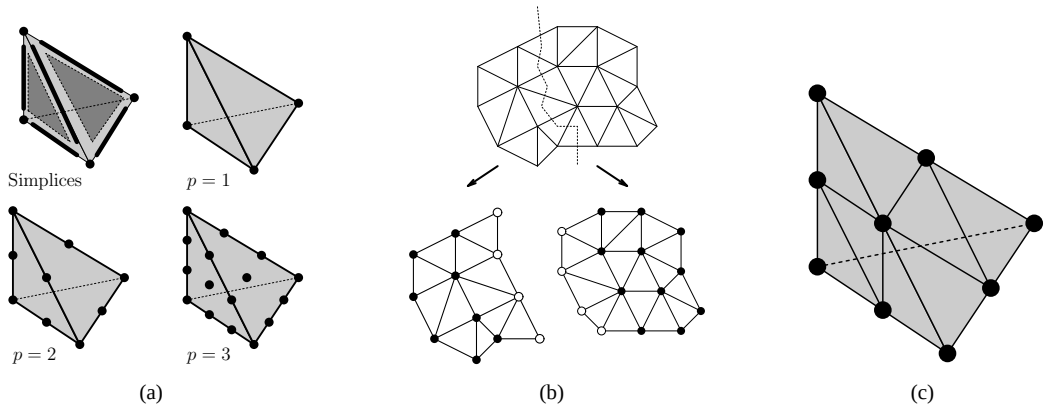


Figure 2. (a)Simplices and structure for FEM polynomial degrees $p = 1, 2, 3$ (b) Decomposition: •=adherent nodes, ○=ghost nodes (c) Mesh refinement.

lar faces (2-simplex) and tetrahedral cells (3-simplex) as illustrated in Fig. 2(a). For a piecewise linear FEM setting where the polynomial degrees are unity, only node and tetrahedra information are required. Additional information are needed for higher order FEM setting which needs higher polynomial degrees. In order to ensure global continuity of the shape functions at incident tetrahedral elements, edge information is required for an FEM setting using a quadratic polynomial degree. In general, an FEM using a polynomial degree p requires geometric information concerning the σ-simplices for $\sigma \in \mathcal{J}_p$ where

$$\mathcal{J}_p := \{\sigma : 0 \leq \sigma \leq \min(p-1, 3)\} \cup \{3\}.$$

Our objective is to decompose the domain $\mathbf{\Omega}$ into N_p (N_p being the number of processors) subdomains so that every processor generates, stores and updates its own information concerning the simplices. It is important to organize synchronous indexations of the simplices at the inter-domain where some σ-simplices are shared by different processors. In particular, for an FEM simulation using polynomials of degree p, every processor manages:

- the σ-simplices which are strictly members of its subdomain where $\sigma \in \mathcal{J}_p$,
- the duplicated σ-simplices at the interface region shared with neighboring subdomains for all $\sigma \in \mathcal{J}_p$,
- the local σ-simplices incident upon every tetrahedron.

As an illustration, we have in Fig. 2(b) a 2D situation where a mesh is decomposed onto two processors for the case of piecewise linear FEM. Our method is featured by the fact that only the initial coarse mesh is stored by the root processor. All subsequent tasks are performed in a distributed manner: mesh refinements, generation and storing of the geometric simplices. In that way, all data are kept distributed while the size of the parallel data is unlimited until the usable memory of the available processors is

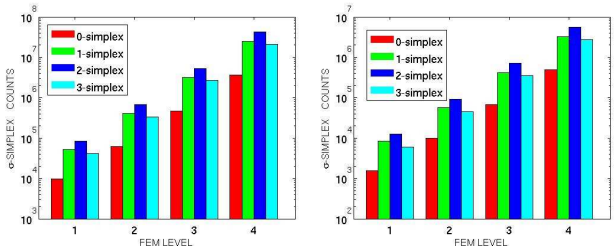


Figure 3. (a) Global counts of the σ-simplices (b) Average counts using eight processors.

reached. The objective of the distribution is to assign to every processor the same amount of computational costs. That is, the processors should generate and store almost the same amount of data. In the case of a chemical simulation using Poisson-Boltzmann equation, there are no hybrid elements. That is, either a tetrahedral element belongs to the solute region $\mathbf{\Omega^u}$ or to the solvent region $\mathbf{\Omega^v}$. As a consequence, the tetrahedral element membership is maintained and updated in all situations such as mesh refinement and interface region tightening.

As soon as the initial biomolecular mesh $\mathcal{M}$ is available on the root processor, it decomposes the domain into N_p subdomains (see Fig. 2(b)) admitting two properties. First, the subdomains are approximately of similar size. Second, the measures of the area shared by adjacent subdomains are as small as possible. As a consequence, the tasks assigned to the processors are balanced: matrix assembly, numerical quadrature for the right hand side and the linear solver. Additionally, the communications between the adjacent processors are minimal. In order to achieve those objectives, one employs a graph which can be assembled in two manners. First, one can use a graph based on the node-edge incidence from the tetrahedral mesh. Another alter-

PROCESSOR	$\sigma=0$		$\sigma=1$		$\sigma=2$		$\sigma=3$	
INDEX	count	ratio	count	ratio	count	ratio	count	ratio
0	220,295	0.9972	1,427,530	1.0267	2,393,740	1.0270	1,186,504	1.0270
1	222,060	1.0052	1,409,054	1.0134	2,361,578	1.0132	1,170,223	1.0129
2	223,107	1.0099	1,410,378	1.0144	2,363,021	1.0138	1,170,681	1.0133
3	220,081	0.9962	1,379,436	0.9921	2,312,412	0.9921	1,146,099	0.9921
4	223,685	1.0125	1,419,116	1.0207	2,377,685	1.0201	1,177,978	1.0197
5	219,031	0.9915	1,365,692	0.9822	2,290,916	0.9829	1,135,979	0.9833
6	219,549	0.9938	1,366,606	0.9829	2,291,826	0.9832	1,136,246	0.9835
7	219,531	0.9937	1,345,249	0.9675	2,255,851	0.9678	1,118,430	0.9681
TOTAL	1,767,339		11,123,061		18,647,029		9,242,140	

Table 1. Balancing of the data on distributed processors. E.g. of a scenario of 8 processors. For crs=12, Level=2

native is to assemble a graph where a graph-node corresponds to a tetrahedron while a graph-edge corresponds to two tetrahedra sharing a common triangular face. One applies to the resulting graph a partitioning algorithm which consists in decomposing the graph into subgraphs of similar size where the number of graph edges to be cut is minimal. For the implementation, we used METIS to achieve that task concerning graph partitioning. Both of the above graph assembly strategies have their own advantages and drawbacks. In fact, since the FEM matrix assembly follows the element-by-element traversal approach, a distribution based on tetrahedral elements enables the processors to have comparable task loads during the FEM matrix assembly. But the sizes of the resulting distributed linear system are not optimally identical. In our case, we have opted to use a nodal distribution which is optimal for a piecewise linear FEM setup. For higher order FEM, a distribution based on nodes is also efficient as we observe in the following data repartition on the different processors. The global number of simplices on refinement levels 1 till 4 are exhibited in Fig. 3(a). By using eight processors, the average numbers of simplices on each refinement level are displayed in Fig. 3(b). The general behaviors of the increase of the number of simplices for the global and local simplices are similar. Apart from the analysis of the average numbers, we examine also the data repartition on all processors. In Tab. 1, we amalgamate the distribution of the data on different processors. We investigate the deviation between the ideal numbers of simplices and their actual numbers. That is observed in the ratio between the average number of the σ-simplices and the number of the ones stored and generated by each processor. It is impossible to obtain the ideal ratio of unity for all simplices. Nevertheless, it can be observed that the proposed method enables a good balance on the processors because all ratios approximate the unity in all treated simplices.

Every σ-simplex is assigned to one and only one supporting processor. Those simplices will be termed the *adherent* simplices with respect to their corresponding processor/subdomain. A σ-simplex s is a *ghost* one w.r.t. a processor P if s is adherent to another processor Q and if s is incident upon a tetrahedron having an adherent simplex belonging to P. In Fig. 2(b), the adherent (*resp.* ghost) nodes are identified by large dots (*resp.* unfilled dots) while the information at the inter-region are duplicated. Every processor stores its own adherent simplices and duplications of ghost simplices from incident subdomains. We denote by $\mathcal{AD}(\sigma)$ and $\mathcal{GH}(\sigma)$ the set of adherent and ghost σ-simplices. Every simplex, whether it be adherent or ghost, is identified by its local index, its adherence flag (to which subdomain it is adherent) and its global mapping (mapping to its global index). A list of global indices in not explicitly stored anywhere. Next, we describe the coherent indexations of simplices on the distributed processors. The

Algorithm: σ-simplex mappings ($\sigma \geq 1$)

Fill global mappings $\mu[i]$ where $i \in \mathcal{AD}(\sigma)$ (adherence)
Assemble $\mathcal{H}$ hash-table(NODE$\longrightarrow \sigma$-simplices)
Assemble $Q := \{r : P_r \in \mathcal{N}\}$
for $r \in Q$:
  Initialize $\beta := \emptyset$
  for $q \in \mathcal{GH}(\sigma)$ (ghost):
    $(n_1, ..., n_{\sigma+1})$ corners nodes of q
    $\beta := \beta \cup \{(n_1, ..., n_{\sigma+1})\}$
  end
  Inter-process: SEND β to processor P_r
end
if (RECEIVE β from a processor P_s)
  Initialize $\gamma := \emptyset$
  for $(n_1, ..., n_{\sigma+1}) \in \beta$:
    Efficient inverse: $GTL(n_i) \rightarrow \ell_i$
    Hash-table look-up: $\mathcal{H}(\ell_i) \rightarrow K_i$ for all $i = 1, ..., \sigma + 1$,
    $w := \bigcap_{i=1}^{\sigma+1} K_i$ and $\gamma := \gamma \cup \{w\}$
  end
  Inter-process: SEND γ to processor P_s
endif

main principle is that the simplices procedure is performed in a distributed manner while still maintaining a coherent global structure. We would like to consider the σ-simplices for $\sigma \geq 1$. Their determination is like in the sequential case which traverses the list of tetrahedra and then traverse the local σ-simplices within each tetrahedron. In the next discussion, we consider the two main difficulties which are the assignment of an adherence flag and that of coherent global mapping. During the adherence flag assignment of the σ-simplex ($\sigma \geq 1$), we assume that all information concerning the nodes are complete whether that be adherent nodes or ghost ones. Consider a σ-simplex such that its nodes are $[n_1, ..., n_{\sigma+1}]$. If all n_i are adherent to the same processor, then we define the adherence flag as that of the processor. Otherwise, we need a convention. For example, the adherence of the simplex is the same as the smallest adherence of the constituting nodes n_i. The global mappings of adherent simplices are achieved completely without interprocessor communications. The assignment of the global index to the adherent simplices is immediate as soon as the adherence information is complete. Only processor p assigns the global indices to the simplices which are adherent to the p-th domain. The main problem is the assignment of the global indices to the ghost simplices because they need to be acquired from appropriate incident processors. We summarize the procedure of achieving that in the presented Algorithm. Inter-processor information dispatching is unavoidable in order that the global mapping is coherent.

For the presented Algorithm, $\mathcal{N}$ stands for the set of processors whose subdomains overlap with the subdomain of the local processor. One needs an inverse mapping which is efficient and which requires only a low memory storage. That is, we suppose we have an injective mapping $\mu : \{0, ..., n\} \longrightarrow \{0, ..., M\}$ such that M is very large. An illustrative scenario is to assume $M \sim n * N_p$ while M is the number of the whole global σ-simplices. One needs to find quickly $\mu^{-1}(x)$ if x is a certain image $x = \mu(y)$. That process is denoted by GTL (global to local) in the displayed Algorithm. The easiest method is to store an array of the inverse of μ. While that is very fast to evaluate and simple to implement, that would require too much memory so that it would limit the potential of the parallel mesh processing. Another approach is to store nothing additional and to traverse the array members of μ as a search for every evaluation of μ^{-1}. That would require no additional memory but the cost of the evaluation is too high. We need an efficient method where only the number of available processors has the limitation. Our assumption is that the sequence of images $\mu(i)$ is given in blocks but they may be unordered. Such an assumption agrees with our whole inter-level constructions. One applies first a preprocessing of an increasing sorting inside each blocks. That preprocessing is very fast because of the availability of QSORT algorithm in all C/C++ release. Storing only the initial value in each block and using linked lists would then facilitate the search. That procedure requires at most $\mathcal{O}(n)$ temporary memory storage while the evaluation speed is fast.

Now, we would like to survey the case of mesh refinements. The presented approach applies to more general cases where only certain tetrahedral elements are subdivided. But in the current paper, only the case of the entire subdivision is considered. One level of refinements consists in uniformly decomposing each tetrahedra into subtetrahedra. That is, one splits every edge by inserting a new node at the midnode. The details of such a local subdivision is illustrated in Fig. 2(c). As in the previous case about simplex treatment, the main problem regarding parallel refinements is again the adherence and the global mappings. The indexations of the old nodes are kept intact. The treatment of global mappings of the newly generated nodes follows the same idea as in the previous σ-simplices with some minor modifications. After refinements, the volumes of the ghost region become tighter. As a consequence, the positions of the tetrahedra with respect to the solute/solvent regions in the biomolecular data need to be updated during the course of the refinement procedure.

4 Outcome of the Parallel Implementation

In this section, we present some practical results of the former parallel approach. In particular, we examine the agreement of the accuracies of the parallel outputs with known theoretical expectations. Let us consider a simulated case where the exact solution is known so that one can compare the computed solution and the exact one. The right hand side is computed by applying the PBE to the exact solution where the domain of simulation is taken as $\mathbf{\Omega} = [0, 1]^3$. We consider a tetrahedral mesh on the unit cube where the step-size along each axis is uniformly of length h. We investigate the errors by comparing the exact solution u and the FEM approximation u_h by using different error gauges. We measure the average errors by means of 100 random points per tetrahedron. In addition, we use the L_∞ error $\|u - u_h\|_\infty = \max_{\mathbf{x}\in\mathbf{\Omega}} |u(\mathbf{x}) - u_h(\mathbf{x})|$. Finally, we evaluate the L_2 error $\|u-u_h\|_2 = \left[\int |u-u_h|^2\right]^{1/2}$. The number of quadrature points for the computation of the integration was intentionally taken very large in order to ensure that the error measurement is not affected by them.

Firstly, we examine the case of piecewise polynomials where the step size h is increased uniformly. An inefficient implementation would provide unexpected results when the parameters $(\varepsilon_u, \varepsilon_v)$ and (κ_u, κ_v) become highly discontinuous. We collect the results of that test in Tab. 2 where one varies the values of $(\varepsilon_u, \varepsilon_v)$ while the values of κ_u and κ_v are fixed. The converse case is shown in Tab. 3. One observes in both cases that the FEM error of our parallel implementation agrees with the theoretical prediction. In fact, the result follows the expected Galerkin error of $\mathcal{O}(h)$ which is found in [12] by using piecewise linear approximation. The discontinuity parameters affect the errors a bit but the convergence behaviors are similar for all situations. We want also to investigate the accuracy of the

$1/h$	$(\varepsilon_u, \varepsilon_v) = (2, 80)$			$(\varepsilon_u, \varepsilon_v) = (2, 40)$			$(\varepsilon_u, \varepsilon_v) = (2, 3)$		
	Avr-ERR	L_∞-ERR	L_2-ERR	Avr-ERR	L_∞-ERR	L_2-ERR	Avr-ERR	L_∞-ERR	L_2-ERR
15	1.35e-03	4.02e-02	6.11e-03	2.42e-03	4.07e-02	8.11e-03	6.87e-03	8.12e-02	7.81e-02
18	9.01e-04	2.89e-02	4.25e-03	1.59e-03	2.87e-02	5.65e-03	4.54e-03	5.58e-02	5.44e-02
21	6.46e-04	2.09e-02	3.12e-03	1.19e-03	2.14e-02	4.15e-03	3.23e-03	4.39e-02	4.00e-02
24	5.57e-04	1.63e-02	2.39e-03	8.31e-04	1.60e-02	3.18e-03	2.44e-03	3.41e-02	3.06e-02
27	4.06e-04	1.29e-02	1.89e-03	7.37e-04	1.28e-02	2.51e-03	1.97e-03	2.71e-02	2.41e-02
30	3.32e-04	1.04e-02	1.53e-03	5.59e-04	1.06e-02	2.03e-03	1.63e-03	2.24e-02	1.95e-02
33	2.77e-04	8.59e-03	1.26e-03	4.65e-04	8.84e-03	1.68e-03	1.40e-03	1.88e-02	1.61e-02
36	2.37e-04	7.44e-03	1.06e-03	3.59e-04	7.10e-03	1.41e-03	1.14e-03	1.60e-02	1.35e-02

Table 2. Accuracy for fixed $(\kappa_u, \kappa_v) = (1, 100)$ using piecewise linear FEM.

$1/h$	$(\kappa_u, \kappa_v) = (1, 100)$			$(\kappa_u, \kappa_v) = (1, 50)$			$(\kappa_u, \kappa_v) = (1, 2)$		
	Avr-ERR	L_∞-ERR	L_2-ERR	Avr-ERR	L_∞-ERR	L_2-ERR	Avr-ERR	L_∞-ERR	L_2-ERR
15	2.42e-03	4.07e-02	8.11e-03	2.25e-03	4.04e-02	8.12e-03	2.33e-03	4.11e-02	8.13e-03
18	1.59e-03	2.87e-02	5.65e-03	1.67e-03	2.87e-02	5.66e-03	1.47e-03	2.86e-02	5.66e-03
21	1.19e-03	2.14e-02	4.15e-03	1.15e-03	2.12e-02	4.16e-03	1.16e-03	2.13e-02	4.16e-03
24	8.31e-04	1.60e-02	3.18e-03	9.23e-04	1.65e-02	3.18e-03	9.18e-04	1.66e-02	3.19e-03
27	7.37e-04	1.28e-02	2.51e-03	7.62e-04	1.31e-02	2.52e-03	7.50e-04	1.29e-02	2.52e-03
30	5.59e-04	1.06e-02	2.03e-03	5.78e-04	1.05e-02	2.04e-03	5.63e-04	1.04e-02	2.04e-03
33	4.65e-04	8.84e-03	1.68e-03	4.56e-04	8.84e-03	1.68e-03	5.34e-04	8.70e-03	1.69e-03
36	3.59e-04	7.10e-03	1.41e-03	3.92e-04	7.45e-03	1.41e-03	3.69e-04	7.40e-03	1.42e-03

Table 3. Accuracy for fixed $(\varepsilon_u, \varepsilon_v) = (2, 40)$ using piecewise linear FEM.

p	$(\varepsilon_u, \varepsilon_v) = (2, 80)$			$(\varepsilon_u, \varepsilon_v) = (2, 40)$			$(\varepsilon_u, \varepsilon_v) = (2, 3)$		
	Avr-ERR	L_∞-ERR	L_2-ERR	Avr-ERR	L_∞-ERR	L_2-ERR	Avr-ERR	L_∞-ERR	L_2-ERR
1	1.98e-03	6.16e-02	9.50e-03	3.46e-03	6.22e-02	1.26e-02	9.87e-03	1.04e-01	1.21e-01
2	1.16e-04	4.06e-03	3.99e-04	2.04e-04	4.34e-03	5.65e-04	6.21e-04	1.42e-02	6.13e-03
3	5.74e-06	4.04e-04	1.79e-05	1.04e-05	5.00e-04	2.70e-05	3.15e-05	1.61e-03	3.12e-04
4	2.48e-07	1.78e-05	7.68e-07	4.65e-07	4.00e-05	1.16e-06	1.34e-06	1.38e-04	1.34e-05
5	8.37e-09	7.51e-07	2.55e-08	1.42e-08	1.62e-06	4.04e-08	4.30e-08	7.26e-06	4.83e-07

Table 4. Accuracy for fixed $(\kappa_u, \kappa_v) = (1, 100)$ using higher order FEM with increasing polynomial degree.

p	$(\kappa_u, \kappa_v) = (1, 100)$			$(\kappa_u, \kappa_v) = (1, 50)$			$(\kappa_u, \kappa_v) = (1, 2)$		
	Avr-ERR	L_∞-ERR	L_2-ERR	Avr-ERR	L_∞-ERR	L_2-ERR	Avr-ERR	L_∞-ERR	L_2-ERR
1	3.46e-03	6.22e-02	1.26e-02	3.31e-03	6.22e-02	1.26e-02	3.61e-03	6.22e-02	1.26e-02
2	2.04e-04	4.34e-03	5.65e-04	1.90e-04	3.99e-03	5.65e-04	1.92e-04	4.16e-03	5.65e-04
3	1.04e-05	5.00e-04	2.70e-05	1.02e-05	4.16e-04	2.70e-05	1.01e-05	3.80e-04	2.70e-05
4	4.65e-07	4.00e-05	1.16e-06	4.38e-07	3.82e-05	1.16e-06	4.17e-07	3.98e-05	1.16e-06
5	1.42e-08	1.62e-06	4.04e-08	1.36e-08	1.71e-06	4.04e-08	1.32e-08	1.71e-06	4.04e-08

Table 5. Accuracy for fixed $(\varepsilon_u, \varepsilon_v) = (2, 40)$ using higher order FEM with increasing polynomial degree.

	nb tetrahedra	D.O.F.	L_∞-ERR	ERR-reduc (∞)	L_2-ERR	ERR-reduc (2)
LEVEL=1 (h)	41,472	9,541	6.22e-02	—	9.50e-03	—
LEVEL=2 ($h/2$)	331,776	62,281	2.59e-02	2.402	3.39e-03	2.801
LEVEL=3 ($h/4$)	2,654,208	463,249	1.20e-02	2.153	1.15e-03	2.948
LEVEL=4 ($h/6$)	21,233,664	3,608,353	4.92e-03	2.455	4.36e-04	2.638
LEVEL=5 ($h/8$)	169,869,312	28,560,961	2.42e-03	2.031	1.68e-04	2.592

Table 6. Piecewise linear: error reductions per level. Inter-level amounts to full subdivision of each tetrahedra

crs	Nb. coarse tetra.	Nb. final tetra.	Distribution	Whole level
6	5,184	21,233,664	0.179 sec	26.969 sec
9	17,496	71,663,616	0.091 sec	92.760 sec
12	41,472	169,869,312	0.199 sec	275.109 sec
15	81,000	331,776,000	0.310 sec	621.030 sec
18	139,968	573,308,928	0.700 sec	1293.839 sec
21	222,264	910,393,344	0.931 sec	2560.308 sec
24	331,776	1,358,954,496	1.447 sec	4417.397 sec
27	472,392	1,934,917,632	2.047 sec	6730.249 sec

Table 7. Size and elapsed runtimne for data distribution: level=5, number of processors=36.

Nb. tetrahedra	D.O.F.	Nb. PCG-iterat.	Residual	Setup	Solve
2,654,208	463,249	129	9.616e-09	0.5228 sec	25.0889 sec
8,957,952	1,547,641	188	9.987e-09	0.8954 sec	42.1407 sec
21,233,664	3,597,985	301	9.070e-09	1.4088 sec	68.5293 sec
41,472,000	7,106,761	320	9.829e-09	2.5147 sec	102.7691 sec
71,663,616	12,080,449	450	9.786e-09	3.9483 sec	186.1989 sec
169,869,312	28,560,961	602	9.500e-09	8.3579 sec	415.1201 sec
241,864,704	41,221,225	572	9.653e-09	10.3053 sec	467.0384 sec

Table 8. Parallel PCG solver using 30 processors to drop the residual error below 1.0e-8. $(\varepsilon_u, \varepsilon_v, \kappa_u, \kappa_v) = (1, 100, 2, 40)$ for piecewise linear.

parallel program for the case of increasing polynomial degrees. In that case, we need sometimes simplices of higher orders as described previously. The various accuracies are gathered in Tab. 4 and Tab. 5 which contain respectively the cases where $(\varepsilon_u, \varepsilon_v)$ and (κ_u, κ_v) are allowed to vary. We notice that the parallel FEM implementation exhibit an exponential convergence which is also expected theoretically. The discontinuity parameters influence the convergence in a minor way but the general behavior remains valid. The following test consists in examining the error reduction for the inter-level computation. In Tab. 6, we collect the results which show the degree of freedom at each level along with the corresponding errors. Every level corresponds to the uniform mesh refinement described in Section 3.2. For the inter-level error, we compute also the error reduction factor which is the ratio between the current error with the previous error on a coarser tetrahedral mesh. The reduction factor of value larger than or equal to 2 is obtained as it is the theoretically predicted [12] value if one applies the refinement levels where the edge lengths are bisected as described in the previous section.

We examine next the number of tetrahedra from the coarse mesh till the 4-th level of refinement. In Tab. 7, we collect the elapsed runtimes for the initial distribution and the whole execution. The former deals with the acquisition of the data, the graph partitioning related to the coarsest mesh by the root processor and the distribution of that to the other processors. The latter consists of the parallel execution of the data mesh: mesh refinements on all levels and the treatment of the higher order simplices. Here, we examine the case where the size of the coarsest data is growing in a uniform manner.

For the linear solver, we apply a PCG (preconditioned conjugate gradient) using least-square preconditioner. The PCG iteration is realized with the ParaSails implementation. The mesh can be entirely unstructured and no grid hierarchies need to be stored at all level. For highly discontinuous coefficients, no information about the mesh is needed to be explicitly extracted. In Tab. 8, we observe some performance of the PCG where the number of the required iterations to drop the residual error below 10^{-8} does not grow too much in comparison to the DOF (degree of freedom). For that displayed results we considered a piecewise linear FEM. One requires some setup preprocessing before the actual PCG iterations. In Tab. 8, one sees also the required runtime for the the setup preprocessing and the solving procedure for the case where the number of tetrahedra ranges from $2,654,208$ to $241,864,704$ in which one employs 30 processors. In that particular simulation, we used the discontinuous parameters $\varepsilon_u = 1$, $\varepsilon_v = 100$, $\kappa_u = 2$ and $\kappa_v = 40$. It is known that the linear system becomes very difficult to solve when the coefficients ε_u, ε_v, κ_u, κ_v are highly discontinuous or when the polynomial degree grows but it is beyond the scope of this article to exhibit the full potential of the PCG solver.

Conclusion

We proposed a method for treating molecular meshes for subsequent use in FEM method. We aimed at an approach which enable higher order FEM. The entities are well distributed between the different processors. The refinement procedures are applied by each processors with small data exchange. Finally, the method has been applied to FEM simulation using linearized Poisson-Boltzmann equation.

References

[1] E. Herrero, J. Gonzlez, R. Canal, Distributed cooperative caching: an energy efficient memory scheme for chip multiprocessors, IEEE Trans. Parallel Distrib. Syst., 23(5), 2012, 853-861.

[2] F. Bodin, Y. Mevel, S. Chauveau, E. Rohou, Porting an ocean code to MPI using TSF, *Proc. 12th International Workshop on Languages and Compilers for Parallel Computing*, California, USA, 1999, 447-450.

[3] H. Borouchaki and P. George, Aspects of 2-D Delaunay mesh generation, *Int. J. Numer. Methods Eng.*, 40, 1997, 1957-1975.

[4] M. Holst, J. McCammon, Z. Yu, Y. Zhou, AND Y. Zhu, Adaptive Finite Element Modeling Techniques for the Poisson-Boltzmann Equation, (Accepted for publication).

[5] H. Harbrecht, M. Randrianarivony, Wavelet BEM on molecular surfaces: Solvent Exclusive Surfaces, Computing, 92(4), 2011, 335-364.

[6] V. Weijo, M. Randrianarivony, H. Harbrecht, L. Frediani, Wavelet formulation of the Polarizable Continuum Model. J. Comput. Chemistry 31(7), 2010, 1469-1477.

[7] M. Randrianarivony, Adaptive discontinuous Galerkin B-spline on parametric geometries, (Accepted for publication).

[8] M. Randrianarivony, G. Brunnett, Preparation of CAD and molecular surfaces for meshfree solvers, Lect. Notes in Comput. Sci. Eng., 65, 2008, 231-245.

[9] M. Randrianarivony, G. Brunnett. Molecular surface decomposition using geometric techniques, *Proc. Conf. Bildverarbeitung für die Medizine*, Berlin, 2008, 197–201.

[10] M. Connolly, Analytical molecular surface calculation, *J. Appl. Cryst.*, 16 , 1983, 548-558.

[11] M. Randrianarivony, Harmonic variation of edge size in meshing CAD geometries from IGES format, Lect. Notes in Comp. Sci, 5102, 2008, 56-65.

[12] D. Bräss, *Finite Elemente. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie* (Berlin: Springer, 2007).