Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2018-0098

Vol. **25**, No. 4, pp. 1235-1258
April 2019

COMPUTATIONAL SOFTWARE

DASHMM Accelerated Adaptive Fast Multipole Poisson-Boltzmann Solver on Distributed Memory Architecture

Bo Zhang[1,*], Jackson DeBuhr[1], Drake Niedzielski[2], Silvio Mayolo[3], Benzhuo Lu[4] and Thomas Sterling[1]

[1] *Center for Research in Extreme Scale Technologies, School of Informatics Computing, and Engineering, Indiana University, Bloomington, IN, 47404, USA.*
[2] *Department of Physics, Rensselaer Polytechnic Institute, Troy, NY, 12180, USA.*
[3] *Department of Computer Science, Tennessee Technological University, Cookeville, TN, 38505, USA.*
[4] *State Key Laboratory of Scientific/Engineering Computing, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, 100190, China.*

Received 9 April 2018; Accepted (in revised version) 25 September 2018

Abstract. We present DAFMPB (DASHMM-accelerated Adaptive Fast Multipole Poisson-Boltzmann solver) for rapid evaluation of the electrostatic potentials and forces, and total solvation-free energy in biomolecular systems modeled by the linearized Poisson-Boltzmann (LPB) equation. DAFMPB first reformulates the LPB into a boundary integral equation and then discretizes it using the node-patch scheme [33]. It solves the resulting linear system using GMRES, where it adopts the DASHMM library [14] to accelerate the matrix-vector multiplication in each iteration. DASHMM is built on top of a global address space allowing the user of DAFMPB to operate on both shared and distributed memory computers with modification of their code. This paper is a brief summary of the program, including the algorithm, implementation, installation and usage.

AMS subject classifications: 45B05, 31C20, 92C05, 68N19

Key words: Poisson-Boltzmann equation, boundary element method, DASHMM, distributed computing.

*Corresponding author. *Email addresses:* zhang416@indiana.edu (B. Zhang), debuhj@gmail.com (J. DeBuhr), niedzd97@gmail.com (D. Niedzielski), sdmayolo42@students.tntech.edu (S. Mayolo), bzlu@lsec.cc.ac.cn (B. Z. Lu), tron@indiana.edu (T. Sterling)

Program Summary

Program title: DAFMPB

Nature of problem: Numerical solution of the linearized Poisson-Boltzmann equation that describes electrostatic interactions of molecular systems in ionic solutions.

Software license: GNU General Public License, version 3

CiCP scientific software URL:

Distribution format: .gz

Programming language(s): C++

Computer platform: x86_64

Operating system: Linux

Compilers: GCC 4.8.4 or newer. icc (tested with 15.0.1)

RAM:

External routines/libraries: HPX-5 version 4.1.0, DASHMM version 1.2.0

Running time:

Restrictions:

Supplementary material and references: `https://github.com/zhang416/dafmpb`

Additional comments:

1 Introduction

The Poisson-Boltzmann (PB) continuum electrostatic model has been adopted in many simulation tools for theoretical studies of electrostatic interactions between biomolecules such as proteins and DNA in aqueous solutions. Various numerical techniques have been developed to solve the PB equations and help elucidate the electrostatic role in many biological processes, such as enzymatic catalysis, molecular recognition and bioregulation. Packages such as DelPhi [27, 28], MEAD [7], UHBD [3, 36], PBEQ [21], ZAP [19], and MIBPB [10] are based on finite-difference methods. Packages such as the Adaptive Poisson-Boltzmann Solver (APBS) [2] are based on finite volume/multigrid methods. In a circumstance where the linearized PB is applicable, the partial differential equations can be reformulated into a set of surface integral equations (IEs) by using Green's theorem and potential theory [8, 12, 16, 17, 22, 26, 32, 33, 35, 44, 49, 51]. The unknowns in the IEs are located on the molecular surface, and the resulting discretized linear system can be solved very efficiently and accurately with certain fast algorithms, such as the fast multipole method (FMM) [8, 32, 51], pre-corrected FFT [26], or tree codes [17]. This strategy has been implemented in the Adaptive Fast Multipole Poisson-Boltzmann (AFMPB) solver [31, 51], the predecessor of DAFMPB, and many other recent works [12, 16, 44], where [12, 16] also utilized GPU acceleration.

There are two earlier versions of AFMPB. The first one was released as a sequential package written in Fortran [31]. The second one was released in 2015 [51] that used Cilk runtime for parallelization on shared memory computers and provided built-in surface mesh generation capability. Over time, the target applications grew larger and required higher accuracy, which meant that it became difficult or impossible to find a shared memory computer with sufficient memory or processing capacity to solve the problems of interest. The goal of the DAFMPB package is to extend AFMPB's operation to distributed memory architectures to handle larger molecules or situations with higher accuracy requirements, and at the same time, allow its users to operate on both shared and distributed memory computers without modifying their code.

To achieve this goal, the Dynamic Adaptive System for Hierarchical Multipole Method (DASHMM) [14, 15] library was adopted as the central driver of the multipole methods used by DAFMPB. DASHMM is fine-grained, data-driven, and has very good strong-scaling performance. It is built on top of the asynchronous many-tasking HPX-5 [23, 25] runtime system, which provides a global address space. As a result, the adoption of DASHMM fulfills the design goal of DAFMPB. In addition to the distribution of the FMM computation, the use of GMRES by DAFMPB needs to work in distributed memory architectures as well. A GMRES implementation closely following the one used in SPARSKIT [1] is provided in DAFMPB.

The organization of this paper is as follows. Section 2 reviews the mathematical models and discretization methods adopted in DAFMPB, where DASHMM can be applied, and how the solvation-free energy is computed. Section 3 describes how DASHMM is integrated into DAFMPB. Section 4 provides installation guide and job examples. Section 5 shows numerical results on accuracy, convergence, and strong scalability. Section 6 concludes the paper.

2 Overview of the DAFMPB solver

The electrostatic force is considered to play an important role in the interactions and dynamics of molecular systems in aqueous solution. In the Poisson equation model, when the charge density that describes the electrostatic effects on the solvent outside the molecules is approximated by a Boltzmann distribution, the continuum nonlinear Poisson-Boltzmann (PB) equation assumes the following form

$$-\nabla\cdot(\epsilon\nabla\phi)+\bar{\kappa}^2\sinh(\phi)=\sum_{k=1}^{M}q_k\delta(r-r_k). \tag{2.1}$$

In the formula, the molecule is represented by M point charges $\{q_k\}$ located at $\{r_k\}$, ϵ is the position-dependent dielectric constant, ϕ is the electrostatic potential at location r, $\bar{\kappa}$ is the modified Debye-Hückel parameter, where $\bar{\kappa}=0$ in the molecule region and $\bar{\kappa}=\sqrt{\epsilon}\kappa$ in the solution region, and κ is the inverse of the Debye-Hückel screening length

determined by the ionic strength of the solution. When the electrostatic potentials are small, the linearized Poisson-Boltzmann (LPB) equation

$$-\nabla\cdot(\epsilon\nabla\phi)+\bar{\kappa}^2\phi=\sum_{k=1}^{M}q_k\delta(r-r_k) \tag{2.2}$$

becomes valid, equipped with the interface conditions $[\phi]=0$ (by the continuity of the potential) and $[\epsilon\frac{\partial\phi}{\partial n}]=0$ (by the conservation of flux). Here, $[\,]$ denotes the jump across the molecular surface and $\frac{\partial}{\partial n}$ is the outward (towards the solvent) normal direction at the surface.

When Green's second identity is applied, traditional boundary integral equations for the linearized PB equation take the form

$$\frac{1}{2}\phi_r^{\text{int}}=\oint_S^{PV}[G_{L,rr'}\frac{\partial\phi_{r'}^{\text{int}}}{\partial n}-\frac{\partial G_{L,rr'}}{\partial n}\phi_{r'}^{\text{int}}]dS_{r'}+\frac{1}{\epsilon_{\text{int}}}\sum_k q_k G_{L,rr_k},\quad r\in S, \tag{2.3}$$

$$\frac{1}{2}\phi_r^{\text{ext}}=\oint_S^{PV}[-G_{Y,rr'}\frac{\partial\phi_{r'}^{\text{ext}}}{\partial n}+\frac{\partial G_{Y,rr'}}{\partial n}\phi_{r'}^{\text{ext}}]dS_{r'},\quad r\in S, \tag{2.4}$$

where Ω is the molecular domain, $S=\partial\Omega$ is its boundary, i.e., solvent-accessible surface, ϕ_r^{int} and ϕ_r^{ext} are the interior and exterior potential at the surface position $r\in\partial\Omega$, respectively, ϵ_{int} is the interior dielectric constant, $r'\in\partial\Omega$ is an arbitrary point on the boundary, n is the outward normal vector at r', and PV represents the principal value of the integral to avoid the singular point when $r'\to r$ in the integral equations. In the formulae, $G_{L,rr'}=\frac{1}{4\pi|r-r'|}$ and $G_{Y,rr'}=\frac{\exp(-\kappa|r-r'|)}{4\pi|r-r'|}$ are the fundamental solutions to the corresponding Poisson and LPB equations, respectively. Integrals of the form $\oint_S^{PV}G_{L,rr'}f(r')dS_{r'}$ and $\oint_S^{PV}G_{Y,rr'}f(r')dS_{r'}$ are called single-layer Laplace and Yukawa potentials and integrals of the form $\oint_S^{PV}\frac{\partial G_{L,rr'}}{\partial n}f(r')dS_{r'}$ and $\oint_S^{PV}\frac{\partial G_{Y,rr'}}{\partial n}f(r')dS_{r'}$ are called double-layer Laplace and Yukawa potentials. These equations can be easily extended to multi-domain systems in which Eq. (2.3) is enforced for each individual domain and the integration domain in Eq. (2.4) includes the collection of all boundaries.

To complete the system, the solutions to the interior (Eq. (2.3)) and exterior (Eq. (2.4)) are matched by the boundary conditions $\phi^{\text{int}}=\phi^{\text{ext}}$ and $\epsilon_{\text{int}}\frac{\partial\phi^{\text{int}}}{\partial n}=\epsilon_{\text{ext}}\frac{\partial\phi^{\text{ext}}}{\partial n}$, where ϵ_{ext} is the exterior (solvent) dielectric constant. Using these conditions, we can define $f=\phi^{\text{ext}}$ and $h=\frac{\partial\phi^{\text{ext}}}{\partial n}$ as the new unknowns and recover other quantities using boundary integrals of f and h. Unfortunately, theoretical analysis shows that the corresponding system for f and h is in general a Fredholm integral equation of the first kind and hence ill-conditioned, i.e., when solved iteratively using Krylov subspace methods, the number of iterations increases with the number of unknowns, and the resulting algorithm becomes inefficient for large systems. Instead of this "direct formulation", Müller [39], Rokhlin [41], Kress [24], introduced a method where the single and double layer potentials are combined in order to derive an optimized second kind Fredholm integral equation. This type

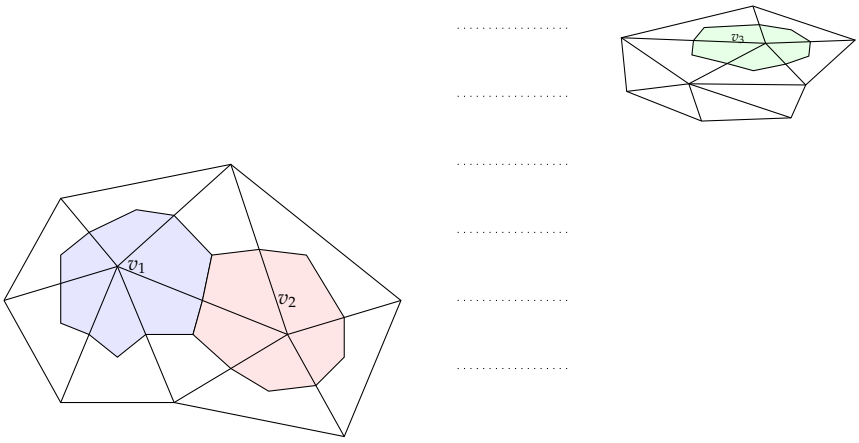


Figure 1: Illustration of the node-patch scheme used to discretize boundary integral equation (2.5) in DAFMPB. For each node v, its patch is enclosed by the edges connecting the centroid of the elements which v belongs to and the midpoints of the edges incident at v. When the nodes are far apart, such as v_1 and v_3, the integrands of the integrations (2.7) are taken as constant on the patch. When the nodes are close, such as v_1 and v_2, the integrations (2.7) are computed directly using detailed information of the patches.

of well-conditioned form also appeared in the work of Juffer et al. [22] and the condition number has been studied in [29]. The well-conditioned form, also called derivative boundary integral form (dBIE) can be obtained by linearly combining the derivative forms of Eqs. (2.3)-(2.4) (another equation is similarly obtained by combining the equation themselves):

$$\begin{aligned}\Big(\frac{1}{2\bar{\epsilon}}+\frac{1}{2}\Big)f_r=&\oint_S^{PV}\left[(G_{L,rr'}-G_{Y,rr'})h_{r'}-\Big(\frac{1}{\bar{\epsilon}}\frac{\partial G_{L,rr'}}{\partial n}-\frac{\partial G_{Y,rr'}}{\partial n}\Big)f_{r'}\right]dS_{r'}\\&+\frac{1}{\epsilon_{\text{ext}}}\sum_k q_k G_{L,rr_k},\quad r\in S,\\ \Big(\frac{1}{2}+\frac{1}{2\bar{\epsilon}}\Big)h_r=&\oint_S^{PV}\left[(\frac{\partial G_{L,rr'}}{\partial n_0}-\frac{1}{\bar{\epsilon}}\frac{\partial G_{Y,rr'}}{\partial n_0})h_{r'}-\frac{1}{\bar{\epsilon}}\Big(\frac{\partial^2 G_{L,rr'}}{\partial n_0\partial n}-\frac{\partial^2 G_{Y,rr'}}{\partial n_0\partial n}\Big)f_{r'}\right]dS_{r'}\\&+\frac{1}{\epsilon_{\text{ext}}}\sum_k q_k\frac{\partial G_{L,rr_k}}{\partial n_0},\quad r\in S,\end{aligned}\tag{2.5}$$

where n_0 is the unit normal vector at point r and $\bar{\epsilon}=\epsilon_{\text{ext}}/\epsilon_{\text{int}}$. DAFMPB and its predecessor AFMPB [31,51] adopt the above formulations. As a Fredholm integral equation of the second kind, the formulation provides the analytic foundation for a well-conditioned system of equations with an adequate discretization method.

DAFMPB and its predecessor AFMPB discretize the integral equation (2.5) using the node-patch scheme [33]. The node-patch around each node v, denoted by ΔS_v, is constructed by connecting the centroid points of the elements that node v belongs to and the midpoints of the edges that are incident to v. Fig. 1 provides an illustration of the node-patch approach on mesh of triangular elements. For instance, the node-patch as-

sociated with v_1 is formed from six surrounding triangular elements. On each patch, the node-patch scheme assumes that the unknowns are constant. This can be considered as a compromise between the traditional constant element method and the linear element method. It draws upon the advantages of both methods with only a slight cost on calculating the node-patch information: (1) the numerical complexity is as simple as in constant element method, while the degree of freedom (DOF) is reduced to about half of that in constant element method (in this aspect, the DOF of node-patch method is as the same as in linear element method) and gaining an overall speedup by 2∼4 times (since the matrix size is quadrupled when DOF is doubled); (2) A similar accuracy as in the linear element method is well maintained. More detailed discussions on this method can be found in [33,50]. Using the node-patch scheme, equation (2.5) can be discretized as

$$\begin{aligned}\left(\frac{1}{2\bar{\epsilon}}+\frac{1}{2}\right)f_i&=\sum_j(A_{ij}h_j-B_{ij}f_j)+\frac{1}{\epsilon_{ext}}\sum_k q_k G_{L,ik},\\ \left(\frac{1}{2\bar{\epsilon}}+\frac{1}{2}\right)h_i&=\sum_j(C_{ij}h_j-D_{ij}f_j)+\frac{1}{\epsilon_{ext}}\sum_k q_k G_{L,ik},\end{aligned}\tag{2.6}$$

where $f_i=f(r_i)$, $h_i=h(r_i)$, the summation on the right-hand side is over all the node-patches $\{\Delta S_j\}$ on the surface, and

$$\begin{aligned}A_{ij}&=\oint_{\Delta S_j}(G_{L,ij}-G_{Y,ij})dS,\\ B_{ij}&=\oint_{\Delta S_j}\left(\frac{1}{\bar{\epsilon}}\frac{\partial G_{L,ij}}{\partial n}-\frac{\partial G_{Y,ij}}{\partial n}\right)dS,\\ C_{ij}&=\oint_{\Delta S_j}\left(\frac{\partial G_{L,ij}}{\partial n_0}-\frac{1}{\bar{\epsilon}}\frac{\partial G_{Y,ij}}{\partial n_0}\right)dS,\\ D_{ij}&=\oint_{\Delta S_j}\frac{1}{\bar{\epsilon}}\left(\frac{\partial^2 G_{L,ij}}{\partial n_0\partial n}-\frac{\partial^2 G_{Y,ij}}{\partial n_0\partial n}\right)dS.\end{aligned}\tag{2.7}$$

In matrix-form, this can be written as

$$\begin{bmatrix}(\frac{1}{2\bar{\epsilon}}+\frac{1}{2})I+B & -A\\ D & (\frac{1}{2\bar{\epsilon}}+\frac{1}{2})I-C\end{bmatrix}\begin{bmatrix}f\\ h\end{bmatrix}=\frac{1}{\epsilon_{ext}}\begin{bmatrix}\sum_k q_k G_{L,ik}\\ \sum_k q_k\frac{\partial G_{L,ik}}{\partial n_0}\end{bmatrix},\tag{2.8}$$

where I is the identity matrix. The discretized system is well-conditioned and can be solved efficiently using Krylov subspace methods. For (2.7), when i and j are far-away, such as v_1 and v_3 in Fig. 1, the integrands in (2.7) are taken as constants; when i and j are close, such as v_1 and v_2 in Fig. 1, each integration is computed directly and the computation requires detailed information on the constitution of the patch. For this reason, coefficients $\{A,B,C,D\}$ for near-field integrations are computed only once and saved in DAFMPB. It is worth mentioning here that the singular integrals in AFMPB and

DAFMPB are ignored. For a traditional constant element BEM for solution of the PBE, the unknowns are considered to be located at the center of each element (triangle here) where it is locally flat and smooth. In this case, all the singular integral can be semi-analytically and accurately calculated, see [35]. Whereas in AFMPB and DAFMPB, we use a node-patch method. The unknown is actually located at each node and the node-patch is not smooth. For a discretized molecular surface, almost every node is like a corner and is geometrically singular. In this case, it is hard to accurately treat singular (strongly singular and hypersingular) integrals. Fortunately, as the Laplace and Yukawa potentials approach the same value when the distance goes to zero, and the two main terms in Eq. (2.5) (the first term in the integral of the first sub-equation, and the second term in the integral of second sub-equation) are subtraction of these two kernels and their derivatives, respectively, we estimate that these two terms are negligibly small in singular integrals where the distance is small. Based on this observation, in our calculations for each node-patch (or node), we actually ignore all the singular integrals within the node-patch. For all the other patches in the near-field list generated by the tree structure for each node, we directly calculate near-singular integrals using the same Gaussian quadrature method. For all far-field patches, the integrals are computed through FMM. All of our numerical experiments in former (e.g. see [32, 34]) and this work demonstrate that such treatments on singular integrals are acceptable and the solver provides sufficiently accurate results based on the accuracy analysis and comparisons with analytical solutions for sphere case and results by other different solvers like APBS (using finite difference method). It is worth noting that avoiding singularity was also an original goal of the Juffer et al.'s work [22] which derived the boundary integral formulation adopted in AFMPB and DAFMPB.

The overall computation flow of DAFMPB can be summarized as follows:

1. Compute the right-hand side of (2.8) using FMM with single-layer and double-layer Laplace potentials.

2. For each pair of nodes i and j whose patch integrations (2.7) are near-field, compute and save A_{ij}, B_{ij}, C_{ij}, and D_{ij}.

3. Solve equation (2.8) using Krylov subspace method. For each matrix-vector multiplication,

   (a) Compute the near-field contribution using the $\{A,B,C,D\}$ coefficients stored in the previous step.

   (b) Compute the far-field contribution with four FMM calls, using single-/double-layer Laplace and Yukawa potentials.

DAFMPB also computes the total free energy by combining the nonpolar and polar (electrostatic) contributions

$$\Delta E = \Delta E_{np} + \Delta E_p. \tag{2.9}$$

The nonpolar term E_{np} includes the energetic cost of cavity formation, solvent rearrangement and solute-solvent dispersion interactions introduced when the uncharged solute is brought from vacuum into the solvent environment. It is computed using an empirical approach that depends on the molecular surface area and volume

$$\Delta E_{np}=\gamma S+pV+b, \tag{2.10}$$

where S and V are the surface area and volume of the cavity created by the molecule, respectively, and γ, p and b are fitted parameters specified through the command line arguments. Parameter γ has the dimensions of a surface tension coefficient, parameter p has the pressure dimensions, and parameter b has the energy dimensions. These parameters depend on the force field and the definition of molecular surface because the atomic radii and charges (from the force field) and the surface definition determine together the boundary between the solvent and the solute and, therefore, the surface area S, volume V, and the polar contributions to free energies. For instance, the particular values $\gamma=0.005\text{kcalmol}^{-1}\text{Å}^{-2}$, $p=0.035\text{kcalmol}^{-1}\text{Å}^{-3}$, and $b=0\text{kcalmol}^{-1}$ are used in [45]. In this paper, we omit the attractive nonpolar solvation interactions, for the same reason given in [45]. We mention that the nonpolar solvation energy expressions similar to Eq. (2.10) can be derived using the scaled particle theory [40]. A more detailed information for the evaluation of these terms is given in [37]. Eq. (2.10) reduces to the popular area-only nonpolar implicit model when $p=0$ [38, 43].

The polar term E_p is determined by the PB solution. Once the electrostatic potential $\phi(r)$ is obtained, the electrostatic contribution is computed by

$$\begin{aligned}\Delta E_p&=\frac{1}{2}\sum_{k=1}^{M}q_k\phi(r_k)=\frac{1}{2}\sum_{k=1}^{M}q_k\oint_S\left(\bar{\epsilon}G_{L,kj}h_j-\frac{\partial G_{L,kj}}{\partial n}f_j\right)dS_j\\&=\frac{1}{2}\sum_{k=1}^{M}q_k\sum_j\left(\bar{\epsilon}G_{L,kj}h_j\Delta S_j-\frac{\partial G_{L,kj}}{\partial n}f_j\Delta S_j\right).\end{aligned} \tag{2.11}$$

3 DAFMPB implementation

There are two main computational modules in DAFMPB: the GMRES module that solves the discretized system (2.8) and the FMM module that accelerates the matrix-vector multiplication used in each GMRES iteration. The FMM module is implemented using the DASHMM (Dynamic Adaptive System for Hierarchical Multipole Method) library. In this section, we first describe the extension made to the DASHMM to implement DAFMPB and then describe the distributed GMRES implementation.

3.1 Multipole computation using DASHMM

3.1.1 DASHMM 1.2.0

DASHMM is an open-source scientific software library that aims to provide an easy-to-use system that can provide scalable, efficient, and unified evaluations of general hierarchical multipole methods on both shared and distributed memory architectures. Here, we briefly describe the latest public release of DASHMM. Readers interested in more implementation details and their execution characteristics are referred to [13–15].

The parallelization strategy employed in DASHMM deviates significantly from conventional practice in many existing MPI+X implementations, which use static partitioning of the global tree structure and bulk-synchronous communication of the locally essential tree [46–48]. Instead, DASHMM leverages the asynchronous multi-tasking HPX-5 [23, 25] runtime system for asynchrony management. HPX-5 defines a broad API that covers most aspects of the system. HPX-5 programs are organized as diffusive, message driven computation, consisting of a large number of lightweight threads and active messages, executed within the context of a global address space, and synchronized through the use of lightweight synchronization objects. The HPX-5 runtime is responsible for managing global allocation, address resolution, data and control dependence, scheduling lightweight threads and managing network traffic.

For general end-science users, the DASHMM APIs are completely independent of HPX-5 and no knowledge of the runtime is required. A basic use can be achieved simply through the `Evaluator` object

```
dashmm::Evaluator<SourceData, TargetData, Expansion, Method>
```

that takes source data, target data, expansion, and method as template parameters. The method class describes which pair of boxes need to interact with each other and the associated translation operator. The expansion class implements various translation operators for a particular interaction type (kernels). DASHMM comes with a set of widely used kernels, including the Laplace, Yukawa, and the low-frequency Helmholtz kernels. It also provides three built-in methods: Barnes-Hut, the classical FMM, and an variant of FMM that uses exponential expansions [20], which is called `FMM97` method in the library. The multipole computation can be simply performed by calling the `evaluate` method of the `Evaluator` object. For instance, the following code performs a computation of the Laplace potentials using the adaptive FMM method.

```
dashmm::Evaluator<SourceData, TargetData,
                  dashmm::Laplace, dashmm::FMM> laplace_fmm;
laplace_fmm.evaluate(source_handle, target_handle, refinement_limit,
                     method, accuracy, kernel_params);
```

Additionally, users of DASHMM can implement their own expansion or special method provided their implementations conform to the public APIs of the built-in expansions and methods, and the implementation needs to be done only sequentially.

Finally, DASHMM makes a distinction between the mathematical concept of an expansion, and the concept used in DASHMM. The mathematical concept is a truncated series of some form (e.g., spherical harmonics) that represents some potential and is referred as a `View` object. The concept of an expansion in DASHMM is wider: each expansion in DASHMM can contain multiple mathematical expansions. In other words, each expansion in DASHMM is a collection of `Views`, or referred as a `ViewSet` object. The views of a `ViewSet` can represent the same potential, each from a different perspective, which is the case for exponential expansions in the `FMM97` method. The views of a `ViewSet` can also represent different potentials. This allows the four FMM kernel evaluations in each iteration step of the DAFMPB solver to be processed concurrently.

3.1.2 Extensions to DASHMM 1.2.0

When solving Eq. (2.8) using Krylov subspace methods, each iteration involves four FMM computations: single-/double-layer Laplace potentials and single-/double-layer Yukawa potentials. The single-layer Laplace and Yukawa potentials are provided by DASHMM as built-in expansions. Therefore, to complete the computation of each step, DAFMPB requires the implementation of double-layer Laplace and Yukawa potentials. Notice that the multipole/local expansion for the double-layer Laplace (Yukawa) potential shares the same form as the single-layer potential. This means, only three operators are needed for each new kernel:

1. `S_to_M` operator that generates a multipole expansion from a set of source points.
2. `S_to_L` operator that generates a local expansion from a set of source points.
3. `L_to_T` operator that evaluates the local expansion at target points.

The other operators such as the `M_to_M` operator that translates the multipole expansion from child to parent can be shared between the single and double layer potentials.

In addition to implementing two new expansion classes, DAFMPB also requires implementing a new method. In adaptive FMM, the surroundings of a box is organized into four categories [9], usually referred as *lists* (see Fig. 2 for a 2D illustration). The definitions of the lists are as follows:

1. $L_1(B_t)$, the list-1 of B_t, is empty if B_t is a non-leaf node. Otherwise, it contains all the leaf source nodes that are adjacent to B_t.
2. $L_2(B_t)$, the list-2 of B_t, is the interaction list of B_t.
3. $L_3(B_t)$, the list-3 of B_t, is empty if B_t is a non-leaf node. Otherwise, it contains all the nodes that are not adjacent to B_t, but whose parents are adjacent to B_t.
4. $L_4(B_t)$, the list-4 of B_t, consists of all the leaf source nodes that are adjacent to B_t's parent, but not to B_t itself.

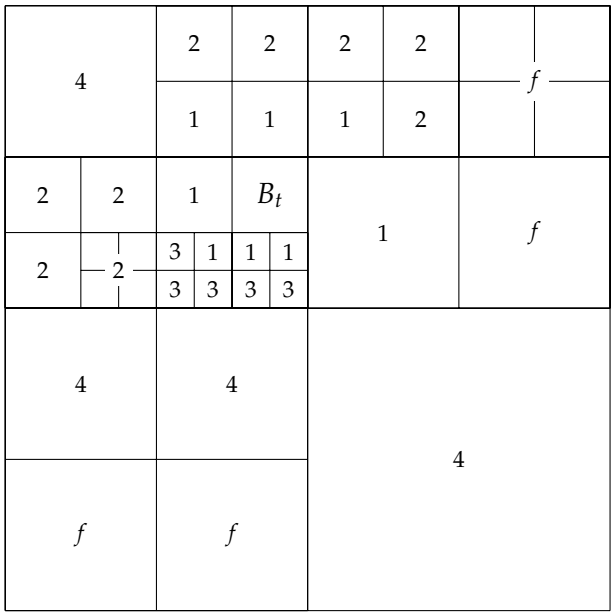


Figure 2: Illustration of the four types of lists associated with a node of the target tree B_t in 2D.

Traditionally, if $B_s \in L_3(B_t)$, then the interaction between B_s and B_t is processed by the `M_to_T` operator, which evaluates the multipole expansion of B_s at each target location contained in B_t. However, to achieve the required accuracy, list-3 boxes in DAFMPB need to be processed in the same way as list-1 boxes, using the `S_to_T` operator. This new method can be implemented with a slight modification of the built-in `FMM97` method: For each list-3 type interaction pair discovered, change its associated operator from the `M_to_T` operator to the `S_to_T` operator.

To facilitate the GMRES solving phase, six new APIs are added to the `Evaluator` object. In version 1.2.0 of DASHMM, the multipole evaluation is done in a monolithic way through the single `evaluate` method. It first constructs the auxiliary structures, such as the dual tree and the directed acyclic graph (DAG), and then evaluates the DAG. This approach provides an easy-to-use and complete evaluation of a given multipole method. However, this one-size-fits-all approach is not well tuned for iterative methods that uses the same DAG multiple times with different input data. To address this, DASHMM now supports evaluation split into phases, including:

- `create_tree`: Partition the source and target points into two trees. The trees can be identical, partially overlapping, or completely disjoint. In DAFMPB, the right-hand side of (2.8) is a case where the sources (atoms inside the molecule) are completely disjoint from the targets (mesh nodes on the molecule surface), and the left-hand side of (2.8) is a case where the sources and targets are the same (mesh nodes on the molecular surface).

- `create_DAG`: This method takes the handle of the tree constructed from `create_tree` and a given multipole `method` object to connect the trees into a DAG.
- `execute_DAG`: This method performs the evaluation of the multipole method.
- `reset_DAG`: This method resets various DASHMM internal control objects. Once complete, the DAG is ready to execute a new round of execution.
- `destroy_DAG`: This method destroys the DAG.
- `destroy_tree`: This method destroys the dual trees.

By separating the evaluation into phases DAFMPB can build the tree and DAG only once, and evaluate that DAG repeatedly. This saves the overhead of building an identical tree and DAG for each iteration.

Another extension implemented in DASHMM for DAFMPB is the `Serializer` object. In (2.8), the near-field computation requires the generation of the $\{A,B,C,D\}$ coefficients. One can first generate these coefficients before starting the iterative solution phase. This synchronization barrier blocks the far-field evaluation of the first matrix-vector multiplication of the iterative solve, and increases the overall execution time. Instead, DAFMPB directly enters the iterative solve phase, generates and saves the $\{A,B,C,D\}$ coefficients for future iterations while making progress on the far-field evaluation. This suggested additional functionality from DASHMM: the detailed patch information associated with each node is needed only in the first iteration and one should avoid communicating unnecessary large messages. The `Serializer` object is introduced to address this issue. It has three member functions:

- `size`: This takes a handle to the object in question and returns the size in bytes of the serialized object.
- `serialize`: This serializes the given object into the buffer provided and then returns the address after the serialized data.
- `deserialize`: This deserializes the given object into the buffer provided and then returns the address after the data used in the deserialization.

Furthermore, DASHMM defines a method, `set_manager`, on its `Array` type that allows the array to update its binding to a `Serializer` during the course of execution. Two `Serializer` objects are defined in DAFMPB: one serializes detailed patch information and is used only in the first iteration, and the other serializes the new input vector generated from the Krylov solver in the successive iterations.

3.2 GMRES

DAFMPB uses GMRES to solve the discretized system (2.8). Notice that the system is a second-kind Fredholm integral equation and it was shown in [29] that the computed condition number of the generated matrix was not large. Together with our past numerical

tests in the previous works, the system has been demonstrated to be well-conditioned. For these reason, the GMRES implementation follows closely to the restarted one in SPARSKIT [1], skipping statements on preconditioners. Notice further that the input to DASHMM in each iteration is the last orthonormal basis formed. Therefore, to reduce unnecessary communications, the Krylov basis are distributed exactly the same way as the mesh nodes. In other words, each `Node` object has a `gmres` member that stores the corresponding Krylov basis components.

4 Software installation and job examples

DAFMPB package depends on two external libraries: DASHMM and HPX-5. Assume the DAFMPB source has been unpacked into the folder `/path/to/dafmpb`. Tarball of version 4.1.0 of HPX-5 can be found in the `/path/to/dafmpb/contrib` folder. DASHMM is automatically downloaded by DAFMPB when the package is built.

Users of DAFMPB must install HPX-5 on their systems first. HPX-5 currently specifies two network interfaces: the ISend/IRecv interface with the MPI transport, and the Put-with-completion (PWC) interface with the Photon transport. The PWC interface supports remote direct memory access. HPX-5 can be built with or without network transports. Assume that you have unpacked the HPX-5 source into the folder `/path/to/hpx` and wish to install the library into `/path/to/install`. The following steps should build and install HPX-5 without network support.

```
cd /path/to/hpx
./configure --prefix=/path/to/install
make
make install
```

To configure HPX-5 with the MPI transport, simply add `--enable-mpi` to the configure line. The configuration will search for the appropriate way to include and link to MPI: (1) HPX-5 will try and see if `mpi.h` and `libmpi.so` are available with no additional flags; (2) HPX-5 will test for an `mpi.h` and `-lmpi` in the current `C_INCLUDE_PATH` and {`LD`}`_LIBRARY_PATH` respectively; and (3) HPX-5 will look for an `ompi pkg-config` package.

To configure HPX-5 with the Photon transport, one adds `--enable-photon` to the configure line. HPX-5 does not provide its own distributed job launcher, so it is necessary to also use either the `--enable-mpi` or `--enable-pmi` option in order to build support for `mpirun` or `aprun` bootstrapping. Note that if you are building with the Photon network, the libraries for the given network interconnect you are targeting need to be present on the build system. The two supported interconnects are InfiniBand (`libverbs` and `librdmacm`) and Cray's GEMINI and ARIES via uGNI (`libugni`). On Cray machines you also need to include the `PHOTON_CARGS=''--enable-ugni''` to the configure line so that Photon builds with uGNI support. Finally, the `--enable-hugetlbfs` option causes

the HPX-5 heap to be mapped with huge pages, which is necessary for larger heaps on some Cray Gemini machines.

Once the HPX-5 system is installed, one needs to modify the following environment variables

```
export PATH=/path/to/install/bin:$PATH
export LD_LIBRARY_PATH=/path/to/install/lib:$LD_LIBRARY_PATH
export PKG_CONFIG_PATH=/path/to/install/lib/pkgconfig:$PKG_CONFIG_PATH
```

and then completes the installation using `cmake` (version 3.4 and above) as follows

```
> mkdir build; cd build
> cmake /path/to/dafmpb
> make
```

The previous commands will automatically download and build the DASHMM library, so no extra steps are required to satisfy that dependency.

The default name of the executable of the package is `dafmpb`. It can be used simply as

```
./dafmpb --pqr-file=FILE
```

which will discretize the molecule using the built-in mesh generation tool and compute the potentials and solvent energy. Other available options to the program can be queried using `./dafmpb --help`, which include

–mesh-format=num Available choices are 0, 1, 2. 0 indicates built-in mesh, 1 indicates MSMS mesh [42], and 2 indicates TMSmesh [11].

–mesh-file=FILE Required if the mesh format is not zero.

–mesh-density=num Specifies mesh density if built-in mesh is selected.

–probe-radius=num Specifies probe radius if built-in mesh is selected.

–dielectric-interior=num Specifies the interior dielectric constant.

–dielectric-exterior=num Specifies the exterior dielectric constant.

–ion-concentration=num Specifies the ionic concentration in mM.

–temperature=num Specifies the temperature.

–surface-tension=num Specifies the surface tension coefficient.

–pressure=num Specifies the pressure.

–accuracy=num Specifies the accuracy of DASHMM, available choices are 3 and 6.

–refine-limit=num Specifies the refinement limit used by the multipole method.

–rel-tolerance=num Specifies relative tolerance of (5.1) for GMRES solver.

–abs-tolerance=num Specifies absolute tolerance of (5.1) for GMRES solver.

–restart=num Specifies the maximum dimension of Krylov space before it restarts.

–max-restart=num Specifies the maximum number of times GMRES can restart.

–log-file=FILE Name of the log file.

–potential-file=FILE Name of the file containing computed potentials on each mesh node.

Next, we use the 3K1Q molecule to be discussed in Section 5 to present some sample job scripts. On a cluster using Slurm workload manager, a job using 512 compute nodes looks like

```
#! /bin/bash -l
#SBATCH -p queue
#SBATCH -N 512
#SBATCH -J jobname
#SBATCH -o output
#SBATCH -e error
#SBATCH -t 00:10:00
srun -n 512 -c 48 ./dafmpb --pqr-file=3k1q.pqr --mesh-format=2 \
--mesh-file=3k1q.off --log-file=3k1q.out512 --accuracy=6 --restart=79 \
--max-restart=5 --hpx-threads=24
```

It is worth noting that each compute node of the above cluster is equipped with two Intel Xeon E5 12-core CPUs with hyperthreading enabled. This means, Slurm sees 48 cores per compute node. The option `-c 48` is to create one MPI process per compute node during the HPX-5 bootstrapping phase. As there are only 24 physical cores on each compute node, the option `--hpx-threads=24` means that after initial bootstrapping, HPX-5 will create only 24 scheduler threads, one for each physical core. If the above cluster were using the PBS workload manager, the script will look like

```
#! /bin/bash -l
#PBS -l nodes=512:ppn=48
#PBS -l walltime=00:10:00
#PBS -q queue
aprun -n 512 -d 48 ./dafmpb ....
```

5 Numerical examples

In this section, we demonstrate the accuracy and parallel efficiency of the DAFMPB package. The accuracy aspect is demonstrated by considering a spherical case with known analytic solution, a comparison against the APBS [6] package for an asymmetric case, and a convergence study using protein fasciculin II. The parallel efficiency aspect is demonstrated by applying DAFMPB to two large molecule systems: an aquareovirus virion 3K1Q and a dengue virus 1K4R. The results reported in this section were collected from a Cray XC30 cluster at Indiana University. Each compute node has two Intel Xeon E5 12-core CPUs and 64 GB of DDR3 RAM. The HPX-5 runtime was configured with the Photon network transport.

The error tolerance for the GMRES solver is chosen to be

$$\epsilon_{rel}\|b\|_2+\epsilon_{abs}, \tag{5.1}$$

where b represents the right-hand side of (2.8), and the relative and absolute tolerance ϵ_{rel} and ϵ_{abs} depend on DASHMM's accuracy, and the molecule system. When DASHMM computes the matrix-vector multiplication with 6-digit relative accuracy, the lower bound for ϵ_{rel} is 10^{-6}. When DASHMM computes the matrix-vector multiplication with 3-digit relative accuracy, the lower bound for ϵ_{rel} is 10^{-3}.

The first test example used an imaginary spherical shaped molecule of 50Å radius carrying 50 elementary charges at the center. The ionic concentration was set to 0 mM. We computed the numerical solution using a surface mesh consisting of 1,310,720 triangle elements and 655,362 nodes. When DASHMM operates with 3-digit relative accuracy, the polar solvation energy computed from the DAFMPB solver is $-4065.42kcal/mol$. When DASHMM operates with 6-digit relative accuracy, the polar solvation energy computed from the DAFMPB solver is $-4057.83kcal/mol$. The analytical solution of the polar solvation energy is $-4045.5878kcal/mol$ and the relative differences computed from DAFMPB are 0.49% and 0.3%, respectively. It is worth mentioning that the reason we chose such a big sphere is to avoid issues in numerical integrations caused by the extremely small triangle elements when using high resolution meshes. In fact, in a separate test, we used a sphere with 1Å radius and 1 elementary charge (this is more suitably mimicking a surface-exposed charge, as did in Altman et al.'s work [4]) and DAFMPB computed a polar solvation energy of $-81.8401kcal/mol$, achieving a 1.1% relative accuracy compared to the analytical solution, when using a much coarser mesh with 642 nodes and 1280 triangle elements. However, when the resolution of the mesh was further increased, the triangle element became too small for numerical integration, and DAFMPB was unable to compute results with better accuracy.

In the second example, we consider a non-symmetric case, similar to the setup in the work of Altman et al. [4]. Specifically, we compute the electrostatic solvation-free energy for a protein-sized sphere of radius 20Å with a charge of $+1e$ placed 2Å inside the surface, using both DAFMPB and another popular package APBS. As shown in Fig. 3,

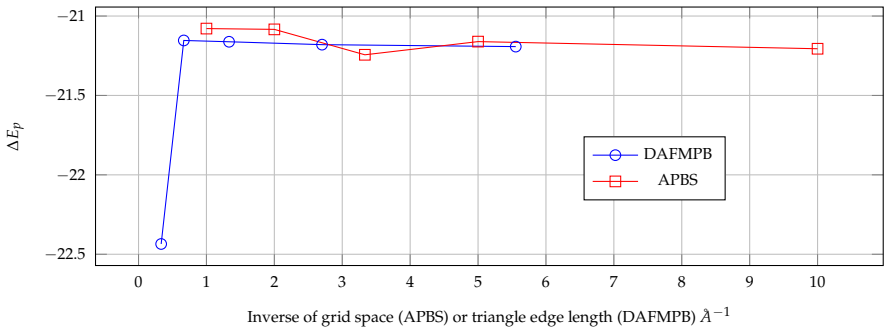


Figure 3: Convergence study in ΔE_p for a protein-sized sphere of radius 20Åwith a charge of $+1e$ placed 2Åinside the surface. For APBS, the grid space ranges from 1Åto 0.1Å. For DAFMPB, the grid space is calculated as the average length of the triangles of the mesh and the range is from 3Åto 0.18Å.

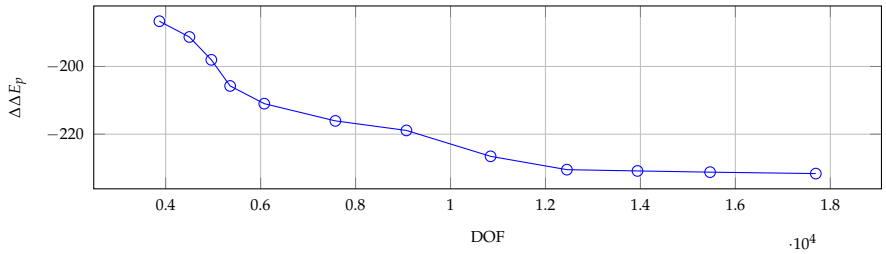


Figure 4: Convergence study in differences of solvation free energies $\Delta\Delta E_p$ of fas2 between the normal state $\Delta_{p,normal}$ and a low-pH state (with five residues neutralized) $\Delta_{p,neutralized}$.

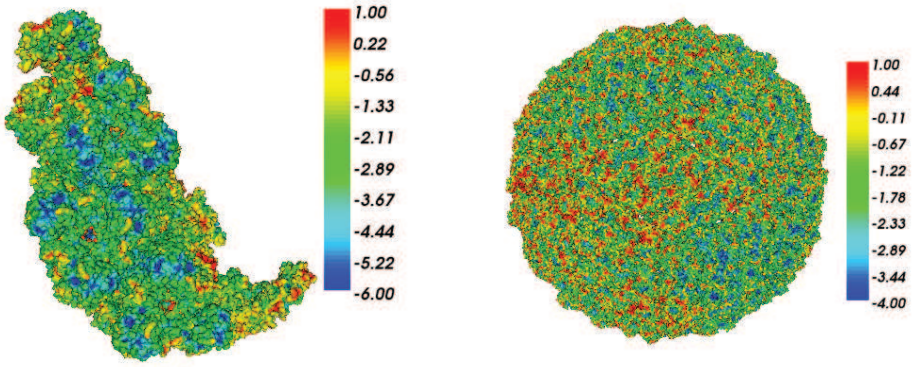
both methods converge quite well and results obtained from the finest meshes are very close, with a relative difference about 0.06%.

In practice, one needs to calculate an energy difference between two states, i.e., $\Delta\Delta E_p$, which is usually small (less than a few tens of $kcal/mol$). This means, a small relative accuracy provided by DAFMPB can lead to a sufficiently good absolute accuracy with respect to $\Delta\Delta E_p$, say, a chemical accuracy $\sim 0.2kcal/mol$. We demonstrate this in the following example using the fasciculinII (fas2), a 68 residue protein. In the tests, we computed the differences in polar solvation free energies of fas2 between two different states, a normal state and a state with solvent-exposed acidic side chains neutralized, mimicking low pH environment (There are 2 GLU and 3 ASP residues whose side chains are neutralized, which makes the total charge increased by $+5e$)

$$\Delta\Delta E_p = \Delta E_{p,neutralized} - \Delta E_{p,normal}.$$

The result is shown in Fig. 4 and it can be seen that $\Delta\Delta E_p$ has a good convergence performance when the number of unknowns (DOFs) increases. Particularly, the variation in $\Delta\Delta E_p$ is within $-0.23kcal/mol$ when the DOF is larger than 12,000.

Surface meshes for the 3K1Q and 1K4R examples used in the parallel efficiency tests



(a) 3K1Q (b) 1K4R

Figure 5: Visualization of the surface potential of the aquareovirus virion (left) and dengue virus (right) systems using VCMM [5]. The surface mesh is generated using TMSmesh and the color bar for the surface potentials is in the unit of $kcal/mol.e$.

were generated using TMSMesh [11,30]. The smaller molecule, 3K1Q, consists of 203,111 atoms and its mesh consists of 7,788,246 triangle elements and 3,888,281 nodes. We were able to test the DAFMPB solver on this molecule with both three and six digit accuracy requirements on DASHMM. At 3-digits of accuracy, the polar energy is $-8.81762\times 10^4 kcal/mol$. At 6-digits of accuracy, the polar energy is $-8.81808\times 10^4 kcal/mol$. The relative difference between the two accuracy requirement is 0.005%, indicating that the lower accuracy input on DASHMM is also acceptable for energy calculations. The larger molecule, 1K4R, consists of 1,082,160 atoms and its mesh consists of 19,502,784 triangle elements and 9,758,426 nodes. For this molecule, we required three digits of accuracy and the polar energy is $-3.999067\times 10^5 kcal/mol$. The output from the DAFMPB package can be visualized by the package VCMM [5]. Fig. 5 shows the visualization of the surface potentials of the two molecule systems. The potential results for the 3K1Q molecule on the left were computed with 6-digit accuracy on DASHMM.

The parallel efficiency tests focused on the strong-scaling performance of the DAFMPB solver. We measured the execution time of the DASHMM evaluation phase and the GMRES phase. Here, the DASHMM phase accumulates the total time spent on matrix-vector multiplication and the GMRES phases accumulates the time spent on generating the basis of the Krylov subspace and estimating the approximation error. The GMRES implementation is based on the modified Gram-Schmidt procedure with reorthogonalization and most of the GMRES time was spent on computing inner products. As each inner product is a global barrier, we also count the number of inner products performed in each test case. For each test case, we started with the smallest number of compute nodes that could complete the computation in specified accuracy requirement and used up to 512 compute nodes for a total of 12,288 cores. We set the GMRES to restart after 80 iterations and allowed the GMRES to restart five times. In all the test cases, DAFMPB converged

Table 1: Strong scaling efficiency of the DASHMM phase for the test cases reported in Fig. 6. The efficiency at p cores is defined as $t_s s/t_p p$, where s is the smallest number of cores used to complete the computation.

Cores	3K1Q, 3-digit	3K1Q, 6-digit	1K4R, 3-digit
192	98%	N/A	N/A
384	97%	N/A	N/A
768	89%	N/A	N/A
1536	57%	65%	95%
3072	66%	98%	87%
6144	45%	69%	60%
12288	27%	42%	45%

well before that limit. For 3K1Q, GMRES took 11 iterations (77 inner products) to converge when DASHMM gave 3-digit accuracy, and 133 iterations (4803 inner products) to converge when DASHMM gave 6-digit accuracy. For 1K4R, GMRES took 10 iterations (66 inner products) to converge. Furthermore, in all test cases, the GMRES phase takes approximately 5% of the total execution and the overall scaling is currently determined by that of DASHMM.

The results are summarized in Fig. 6 and Table 1. From the figure, one notices that the results for the GMRES phase depend on how DASHMM distributes the data. For instance, DASHMM didn't do an optimal job distributing the data at 1536 cores for 3K1Q because the molecular surface is highly irregular (Fig. 5). However, DASHMM distributed the data very well at 3072 cores as the strong-scaling efficiency for GMRES is almost 100%. Nonetheless, each inner product is a global reduction and as the number of compute nodes increases the computation will become insufficient to hide the network communication latency. This explains why all the GMRES curves flatten out at 12288 cores. From the table, one notices that the scaling efficiency is very good up to 768 cores for 3K1Q at lower accuracy requirement and then decays rapidly. For DASHMM, an input size around 4 million points is very small to strong scale beyond 1000 cores. In fact, each single matrix-vector of this size can be completed in a single compute node. The reason we ran the example in distributed memory regime is to have access to sufficient memory space to store the basis of the Krylov subspace. For higher accuracy requirement or large molecule, the scaling efficiency can stay above 60% up to 6144 cores, which is consistent with the ones reported in [14]. However, the efficiency at 12288 cores over 512 compute nodes are much worse, and the cause is very different from the execution characteristic analysis reported in [13].

The inferior scaling performance at 12288 cores is caused by the fact that DAFMPB does not distinguish types 1 and 3 lists in the adaptive FMM (see Section 3.1). In the adaptive FMM, if a source box B_s is in list-3 for target box B_t, their interaction is handled by the `M_to_T` operator. If B_s and B_t are on different compute nodes, one needs to communicate the multipole expansion of B_s and the `M_to_T` operator has enough floating point

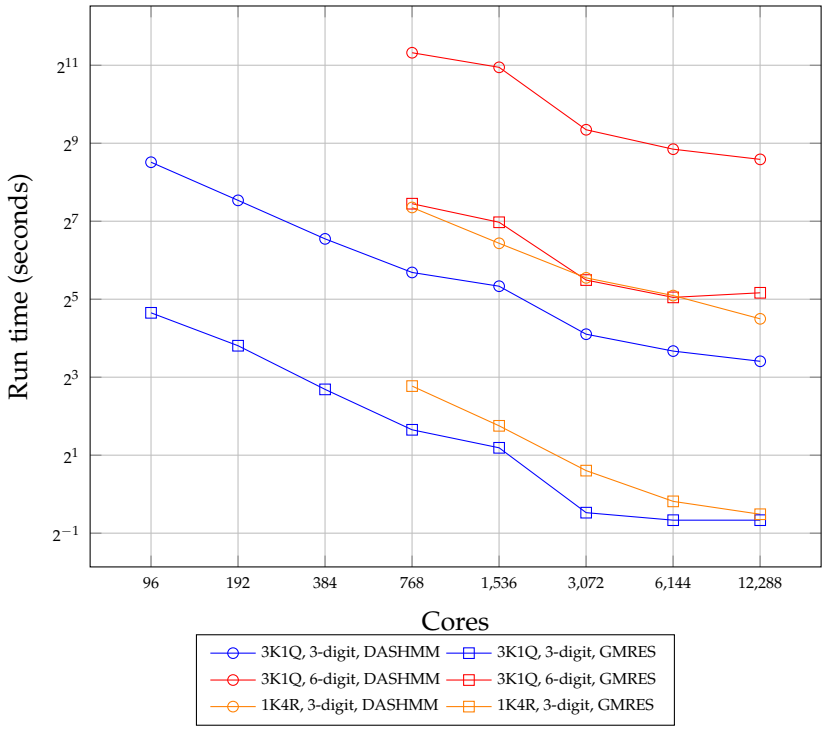


Figure 6: Strong-scaling performance of DASHMM and GMRES phases of DAFMPB on a Cray XC30 cluster. Each compute node has 24 cores and 64 GB of RAM. Each test case started with the smallest number of compute nodes that could complete the computation and used up to 512 compute nodes for a total of 12288 cores.

operations (such as spherical harmonic evaluations) to amortize the communication cost. In DAFMPB, this interaction is handled by the `S_to_T` operator and B_s needs to send "particles" (component of the Krylov basis) to B_t. First, B_s can be a nonleaf box which contains many particles, making the message much larger. Second, once the $\{A,B,C,D\}$ coefficients are computed, for each particle information received, B_t simply does four multiplications, which is not sufficient to amortize the communication cost. The team is currently working on extending the library to heterogeneous architectures. When complete, offloading the near-field interaction should be able to improve the scaling performance.

Finally, we point out that one often chooses restarted GMRES due to the memory limitation on storing the Krylov basis and it often takes more iterations to converge when GMRES restarts. When there are more resources available, one can afford not to restart GMRES and this could lead to shorter time-to-completion. For the 3K1Q example at 6-digit accuracy, we set GMRES to restart at 140 iterations and GMRES actually converged at iteration 89 and the execution time is 30% faster. The polar energy is $-8.83353\times10^4 kcal/mol$. Compared with the one obtained when GMRES restarted, the relative difference is less than 0.17%.

6 Conclusion

We have presented the DAFMPB package for computing electrostatic properties and solvation energies of biomolecular systems. DAFMPB adopted DASHMM as its main computational engine for the FMM solver. On the one hand, DASHMM leverages the global address space of the HPX-5 runtime system, which allows the users of DAFMPB to operate on both shared and distributed memory computers without any modification of their code. On the other hand, the adoption of DASHMM also revealed several issues requiring further study. A better treatment of the near-field interaction is needed to recover the scaling demonstrated by DASHMM in other contexts. One possible approach would be the use of accelerators for near field interactions. The current implementation of the GMRES solver follows closely the one in SPARSKIT. The modified Gram-Schmidt procedure involves many global synchronizations. Given the current breakdown of execution time devoted to FMM and GMRES, it should be possible to hide this synchronization overhead within the matrix-vector multiplication using strategies mentioned in [18] and references therein.

Acknowledgments

Author Lu acknowledges the support of Science Challenge Project (No. TZ2016003), the National Key Research and Development Program of China (Grant No. 2016YFB0201304), and China NSF (NSFC 21573274, 11771435). Authors Zhang, DeBuhr, and Sterling were supported in part by National Science Foundation grant number ACI-1440396. Authors Niedzielski and Mayolo gratefully acknowledge the support of the National Science Foundation's REU program. This research was supported in part by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute.

References

[1] `http://www-users.cs.umn.edu/~saad/software/`.
[2] APBS. http://www.poissonboltzmann.org/apbs.
[3] UHBD. http://mccammon.ucsd.edu/uhbd.html.
[4] M. D. Altman, J. P. Bardhan, J. K. White, and B. Tidor. Accurate solution of multi-region continuum biomolecule electrostatic problems using the linearized poisson-boltzmann equation with curved boundary elements. *J. Comput. Chem.*, 30:132–153, 2009.
[5] S. Bai and B. Lu. VCMM: A visual tool for continuum molecular modeling. *J. Mol. Graphics Modell.*, 50:44–49, 2014.
[6] N. A. Baker, D. Sept, S. Joseph, and M. J. Holst. Electrostatics of nanosystems: Application to microtubules and the ribosome. *Proc. Natl. Acad. Sci. U.S.A.*, 98:10037–10041, 2001.
[7] D. Bashford. An object-oriented programming suite for electrostatic effects in biological molecules. *Lecture Notes in Computer Science*, 1343:233–240, 1997.
[8] A. H. Boschitsch, M. O. Fenley, and H. X. Zhou. Fast boundary element method for the linear Poisson-Boltzmann equation. *J. Phys. Chem. B*, 106:2741–2754, 2002.

[9] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM J. Sci. Stat. Comp.*, 9:669–686, 1988.
[10] D. Chen, Z. Chen, C. Chen, W. Geng, and G. Wei. MIBPB: A software package for electrostatic analysis. *J. Comput. Chem.*, 32:756–770, 2011.
[11] M. Chen and B. Lu. TMSmesh: A robust method for molecular surface mesh generation using a trace technique. *J. Chem. Theory Comput.*, 7:203–212, 2011.
[12] C. D. Cooper, J. P. Bardhan, and L. A. Barba. A biomolecular electrostatics solver using Python, GPUs and boundary elements that can handle solvent-filled cavities and Stern layers. *Comput. Phys. Commun.*, 185:720–729, 2014.
[13] J. DeBuhr, B. Zhang, and L. D'Alessandro. Scalable hierarchical multipole methods using an asynchronous many-tasking runtime system. In *Parallel and Distributed Processing Symposium Workshops*, pages 1226–1234, 2017.
[14] J. DeBuhr, B. Zhang, and T. Sterling. Revision of DASHMM: Dynamic Adaptive System for Hierarchical Multipole Mthods. *Comm. Comput. Phys.*, 2017.
[15] J. DeBuhr, B. Zhang, A. Tsueda, V. Tilstra-Smith, and T. Sterling. DASHMM: Dynamic Adaptive System for Hierarchical Multipole Methods. *Communications in Computational Physics*, 20:1106–1126, October 2016.
[16] W. Geng and F. Jacob. A GPU-accelerated direct-sum boundary integral Poisson-Boltzmann solver. *Comput. Phys. Commun.*, 184:1490–1496, 2013.
[17] W. Geng and R. Krasny. A treecode-accelerated boundary integral Poisson-Boltzmann solver for electrostatics of solvated biomolecules. *J. Comput. Phys.*, 247:62–78, 2013.
[18] P. Ghysels, T. J. Ashby, K. Meerbergen, and W. Vanroose. Hiding global communication latency in the GMRES algorithm on massively parallel machines. *SIAM. J. Sci. Comput.*, 35:C48–C71, 2013.
[19] J. A. Grant, B. T. Pickup, and A. Nicholls. A smooth permittivity function for Poisson-Boltzmann solvation methods. *J. Comput. Chem.*, 22:608–640, 2001.
[20] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numer.*, 6:229–269, 1997.
[21] S. Jo, M. Vargyas, J. Vasko-Szedlar, B. Roux, and V. Im. PBEQ-solver for online visualization of electrostatic potential of biomolecules. *Nucleic Acids Research*, 36:270–275, 2008.
[22] A. H. Juffer, E. F. F. Botta, B. A. M. Vankeulen, A. Vanderploeg, and H. J. C. Berendsen. The electric potential of a macromolecule in a solvent: a fundamental approach. *J. Comput. Phys.*, 97:144–171, 1991.
[23] Ezra Kissel and Martin Swany. Photon: Remote memory access middleware for high-performance runtime systems. In *In Proceedings of the 1st Emerging Parallel and Distributed Runtime Systems and Middleware (IPDRM) Workshop*, 2016.
[24] R. Kress and G.F. Roach. Transmission problems for the helmholtz equation. *J. Math. Phys.*, 19:1433–1437, 1978.
[25] Abhishek Kulkarni, Luke Dalessandro, Ezra Kissel, Andrew Lumsdaine, Thomas Sterling, and Martin Swany. Network-managed virtual global address space for message-driven runtimes. In *Proceedings of the 25th International Symposium on High Performance Parallel and Distributed Computing (HPDC 2016)*, 2016.
[26] S. S. Kuo, M. D. Altman, J. P. Bardhan, B. Tidor, and J. K. White. Fast methods for simulation of biomolecule electrostatics. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design*, pages 466–473, 2002.
[27] C. Li, L. Li, J. Zhang, and E. Alexov. Highly efficient and exact method for parallelization of grid-based algorithms and its implementation in DelPhi. *J. Comput. Chem.*, 33:1960–1966,

2012.

[28] C. Li, M. Petukh, L. Li, and E. Alexov. Continuous development of schemes for parallel computing of the electrostatics in biological systems: implementation in DelPhi. *J. Comput. Chem.*, 34:1949–1960, 2013.

[29] J. Liang and S. Subramaniam. Computation of molecular electrostatics with boundary element methods. *Biophys. J.*, 73:1830–1841, 1997.

[30] T. Liu, M. X. Chen, and B. Z. Lu. Efficient and qualified mesh generation for Gaussian molecular surface using adaptive partition and piecewise polynomial approximation. *SIAM J. Sci. Comput.*, 40:B507–B527, 2018.

[31] B. Lu, X. Cheng, J. Huang, and J. A. McCammon. AFMPB: Adaptive Fast Multipole Poisson-Boltzmann Solver. *Comput. Phys. Commun.*, 181:1150–1160, 2010.

[32] B. Lu, X. Cheng, and J. A. McCammon. New-version-fast-multipole-method accelerated electrostatic interactions in biomolecular systems. *J. Comput. Phys.*, 226:1348–1366, 2007.

[33] B. Lu and J. A. McCammon. Improved boundary element methods for Poisson-Boltzmann electrostatic potential and force calculations. *J. Chem. Theory Comput.*, 3:1134–1142, 2007.

[34] B. Lu, D. Zhang, and J. A. McCammon. Computation of electrostatic forces between solvated molecules determined by the Poisson-Boltzmann equation using a boundary element method. *J. Chem. Phys.*, 122:214102, 2005.

[35] B. Z. Lu, X. L. Cheng, T. J. Hou, and J. A. McCammon. Calculation of the Maxwell stress tensor and the Poisson-Boltzmann force on a solvated molecular surface using hypersingular boundary integrals. *J. Chem. Phys.*, 123:084904, 2005.

[36] J. D. Madura, J. M. Briggs, R. C. Wade, M. E. Davis, B. A. Luty, A. Ilin, J. Antosiewicz, M. K. Gilson, B. Bagheri, L. R. Scott, and J. A. McCammon. Electrostatics and diffusion of molecules in solution-simulation with the University of Houston Brownian Dynamics Program. *Comput. Phys. Commun.*, 91:57–95, 1995.

[37] A. V. Marenich, C. J. Cramer, and D. G. Truhiar. Universal solvation model based on solute electron density and on a continuum model of the solvent defined by the bulk dielectric constant and atomic surface tensions. *J. Phys. Chem. B*, 113:6378–6396, 2009.

[38] I. Massova and P. A. Kollman. Combined molecular mechanical and continuum solvent approach (nm-pbsa/gbsa) to predict ligand binding. *Perspect. Drug Discovery Des.*, 18:113–135, 2000.

[39] C. Müller. *Foundations of the Mathematical Theory of Electromagnetic Waves*. Springer-Verlag, 1969.

[40] R. A. Pierotti. A scaled particle theory of aqueous and nonaqueous solutions. *Chem. Rev.*, 76:717–726, 1976.

[41] V. Rokhlin. Solution of acoustic scattering problems by means of second kind integral equations. *Wave Motion*, 5:257–272, 1983.

[42] M. F. Scanner, A. J. Olson, and J. C. Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38:305–320, 1996.

[43] K. A. Sharp, A. Nicholls, R. F. Fine, and B. Honig. Reconciling the magnitude of the microscopic and macroscopic hydrophobic effects. *Science*, 252:106–109, 1991.

[44] A. M. Tabrizi, S. Goossens, A. M. Rahimi, C. D. Cooper, M. G. Knepley, and J. P. Bardhan. Extending the Solvation-Layer Interface Condition Continuum Electrostatic Model to a Linearized Poisson-Boltzmann Solvent. *J. Chem. Theory Comput.*, 13:2897–2914, 2017.

[45] J. A. Wagoner and N. A. Baker. Assessing implicit models for nonpolar mean solvation forces: The importance of dispersion and volume terms. *Proc. Natl. Acad. Sci.*, 103:8331–8336, 2006.

[46] M. Warren and J. Salmon. Astrophysical n-body simulation using hierarchical tree data structures. In *SC 92': Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, 1992.
[47] L. Ying, G. Biros, D. Zorin, and H. Langston. A new parallel kernel-independent fast multipole method. In *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, 2003.
[48] R. Yokota, J. P. Bardhan, M. G. Knepley, L. A. Barba, and T. Hamada. Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns. *Comput. Phys. Commun.*, 182:1272–1283, 2011.
[49] R. J. Zauhar and R. S. Morgan. A new method for computing the macromolecular electric potential. *J. Mol. Biol.*, 186:815–820, 1985.
[50] B. Zhang, B. Lu, X. Cheng, J. Huang, N. P. Pitsianis, X. Sun, and J. A. McCammon. Mathematical and numerical aspects of the adaptive fast multipole poisson-boltzmann solver. *Commun. Comput. Phys.*, 13:107–128, 2013.
[51] B. Zhang, B. Peng, J. Huang, N. P. Pitsianis, X. Sun, and B. Lu. Parallel AFMPB solver with automatic surface meshing for calculation of molecular solvation free energy. *Comput. Phys. Commun.*, 190:173–181, 2015.