The Journal of Chemical Physics

SOFTWARE

pubs.aip.org/aip/jcp

apoCHARMM: High-performance molecular dynamics simulations on GPUs for advanced simulation methods

Cite as: J. Chem. Phys. **162**, 182501 (2025); doi: 10.1063/5.0264937
Submitted: 12 February 2025 • Accepted: 21 April 2025 •
Published Online: 9 May 2025

View Online Export Citation CrossMark

Samarjeet Prasad,[1,a)] Felix Aviat,[1,2] James E. Gonzales II,[1,3] and Bernard R. Brooks[1]

AFFILIATIONS

[1] Laboratory of Computational Biology, National Heart, Lung, Blood Institute, National Institutes of Health, Bethesda, Maryland 30105, USA

[2] Qubit Pharmaceuticals, Paris, France

[3] Department of Biomedical Engineering, Texas A & M University, College Station, Texas 77843, USA

[a)] **Author to whom correspondence should be addressed:** samar.samarjeet@nih.gov

ABSTRACT

We present apoCHARMM, a high-performance molecular dynamics (MD) engine optimized for graphics processing unit (GPU) architectures, designed to accelerate the simulation of complex molecular systems. The distinctive features of apoCHARMM include single-GPU support for multiple Hamiltonians, computation of a full virial tensor for each Hamiltonian, and full support for orthorhombic periodic systems in both $P1$ and $P2_1$ space groups. Multiple Hamiltonians on a single GPU permit rapid single-GPU multi-dimensional replica exchange methods, multi-state enveloping distribution sampling methods, and several efficient free energy methods where efficiency is gained by eliminating post-processing requirements. The combination of these capabilities enables constant-pH molecular dynamics in explicit solvent with enveloping distribution sampling, where Hamiltonian replica exchange can be performed on a single GPU with minimal host-GPU memory transfers. A full atomic virial tensor allows support for many different pressure, surface tension, and temperature ensembles. Support for orthorhombic $P2_1$ systems allows for the simulation of lipid bilayers, where the two leaflets have equalized chemical potentials. apoCHARMM uses CUDA and modern C++ to enable efficient computation of energy, force, restraint, constraint, and integration calculations directly on the GPU. This GPU-exclusive design focus minimizes host-GPU memory transfers, ensuring optimal performance during simulations, with such transfers occurring only during logging or trajectory saving. Benchmark tests demonstrate that apoCHARMM achieves competitive or superior performance when compared to other GPU-based MD engines, positioning it as a versatile and useful tool for the molecular dynamics community.

I. INTRODUCTION

Molecular dynamics (MD) simulations have become an essential tool for understanding biomolecular systems, offering insights into molecular behavior, conformational dynamics, and energetics at atomic resolution. Recent decades have witnessed remarkable advances in both simulation algorithms and hardware optimizations, such as application-specific integrated circuits (ASICs) in Anton machines[1–3] and general-purpose graphics processing units (GPUs).[4–17] Advances in computational power, particularly with the advent of GPUs, have revolutionized MD simulations, enabling studies of larger systems over longer timescales.[18,19] Traditional MD engines, while powerful, often struggle to fully leverage GPU capabilities, face limitations in ensemble variety, or lack modularity, which can restrict flexibility for customized applications.

The CHARMM MD package has been one of the premier packages in the field of biomolecular simulations.[20–22] However, many of the important features of CHARMM are not available in other GPU-accelerated MD packages. These include a full atomic virial-based Langevin piston barostat, $P2_1$ periodic boundary condition for simulation of bilayers to allow exchange of lipids between the two layers, enveloping distribution sampling, and others. To address these challenges, we introduce apoCHARMM, a novel, GPU-optimized MD engine that provides robust, flexible simulation capabilities, designed to cater specifically to the needs of modern computational researchers. apoCHARMM is built to utilize the high parallel processing capabilities of GPUs, significantly enhancing computational performance while maintaining accuracy. apoCHARMM includes full atomic virial calculations, which are crucial for accurate pressure assessments across different thermodynamic ensembles (in particular NPT, NPAT, NPγT). By supporting multiple ensembles, apoCHARMM allows users to study a wide range of conditions and closely replicate experimental setups. This capability is critical for applications in biomolecular research, where different physiological conditions or experimental setups require specific ensembles to yield meaningful insights.

One of the primary strengths of apoCHARMM is the variety of schemes of free energy calculations that have been implemented, which are essential for understanding processes like ligand binding, protein folding, and conformational changes. The engine has been designed to handle free energy methodologies, providing the precision and flexibility needed for advanced computational studies. Its capacity to conduct rigorous free energy calculations makes apoCHARMM particularly advantageous for drug discovery and materials science, where accurate energetic profiles are invaluable.

The design of apoCHARMM emphasizes modularity, ensuring that the package is easily extensible for future developments or customizations. This allows researchers to add new functionality without modifying the core codebase, making it ideal for those who wish to experiment with novel algorithms or simulation methodologies. Such flexibility is highly valuable in the rapidly evolving field of computational molecular sciences, where researchers continuously seek to push the boundaries of simulation capabilities.

At its core, apoCHARMM is built on a high-performance back-end written in CUDA and C++, allowing it to capitalize on GPU acceleration. By employing the CUDA programming model, apoCHARMM achieves optimized parallelism, which is critical for processing the extensive calculations inherent to MD simulations. In addition, a Python interface powered by pybind11 enhances accessibility, allowing users to run simulations and interact with apoCHARMM through Python scripts. This dual-layered design—CUDA for speed and Python for ease of use—creates an ideal balance between performance and accessibility, catering to users with varying levels of programming expertise.

Finally, apoCHARMM's development follows the principles of Test-Driven Development (TDD), ensuring a robust, reliable codebase where functionalities are validated continuously through rigorous testing. This approach minimizes bugs, facilitates smoother integration of new features, and ensures high-quality code throughout the development lifecycle. By combining performance, accuracy, modularity, and usability, apoCHARMM represents a next-generation MD engine that addresses the current limitations in GPU-based MD simulations, poised to support researchers in tackling increasingly complex questions in molecular science.

In this paper, we present the development, optimization, and performance benchmarks of apoCHARMM, illustrating its potential as a next-generation MD engine. Through a series of benchmark tests, we demonstrate apoCHARMM's performance advantages in calculating the full virial and simulating various ensembles. We also provide case studies that highlight apoCHARMM's capabilities in simulating relevant systems with high accuracy and computational efficiency.

II. MAIN FEATURES

apoCHARMM is an open-source package that has been developed specifically to support some of the distinctive methods of CHARMM[22] at the speeds provided by modern GPU architectures. apoCHARMM has several unique features that are generally not available in other GPU-based MD engines. In particular, apoCHARMM supports the following:

- A complete analytic virial tensor.
- Multiple Hamiltonians at a single time step.
- Innovative crystal symmetries (e.g., $P2_1$).

A complete virial tensor has enabled the implementation of the Langevin piston algorithms[23] for constant pressure or constant surface tension ensembles. The support for multiple Hamiltonians or protein structure files (PSFs) simultaneously allows many free energy methods, such as Multistate Bennett Acceptance Ratio (MBAR),[24] to be run without the need for trajectory storage and slow post-processing. It also allows the Enveloping Distribution Sampling (EDS)-based methods[25–27] for free energies and efficient state-based constant-pH molecular dynamics.[28,29] The upper limit on the number of simultaneously handled structures is defined by the specific hardware being used. Support for $P2_1$ crystal symmetry[30,31] allows lipid bilayer systems to be simulated where there is no chemical potential mismatch between upper and lower leaflets, which can be very useful for membrane insertions.[32] The algorithms for sampling the different ensembles in apoCHARMM are listed in Table I.

In this section, we will introduce the various features and capabilities of apoCHARMM. Details on implementation and code design choices can be found in Sec. III.

A. Virial calculation

Computing the pressure accurately requires the calculation of the internal virial through the commonly used virial equation:

$$P = \frac{Nk_bT}{V} + \frac{1}{3V}\sum_i \mathbf{f_i r_i},$$

TABLE I. Available features in apoCHARMM.

Feature	Algorithms
Energy minimization	Steepest descent, ABNR, L-BFGS
NVE integration	Leapfrog, velocity verlet
NVT thermostat	Nose–Hoover, Langevin
NPT barostat	Langevin-piston

where k_b is the Boltzmann constant, $\mathbf{r_i}$ is the position of the ith atom, and $\mathbf{f_i}$ is the force on the atom i due to all the other atoms in the system. However, for periodic systems with pairwise interactions, the actual equation is given by

$$P = \frac{Nk_bT}{V} + \frac{1}{6V}\sum_{ij,i<j} \mathbf{f_{ij}}\mathbf{r_{ij}},$$

where $\mathbf{f_{ij}}$ is the force between atoms i and j, and $\mathbf{r_{ij}} = \mathbf{r_i} - \mathbf{r_j}$.[33] This implies that an interaction between a particle and an image of another particle needs to be calculated if the image is closer, as per the minimum image convention. Hence, the virial contribution has to be calculated in the inner loop during the calculation of the pairwise force rather than after the forces on the atoms have been calculated. The thermodynamic pressure of the system is defined as the time average of the instantaneous pressure $P(t)$. We calculate the full atomic virial since it is needed for several methods such as the Langevin piston barostat, Martyna–Tobias–Klein (MTK) barostat, and calculation of viscosity.

B. Integrators

A number of different integrators have been implemented in apoCHARMM to enable sampling across different ensembles and optimizing performance for various types of molecular simulations. For simulations in the microcanonical (NVE) ensemble, both velocity-Verlet and leap-frog integrators are available, providing energy-conserving dynamics suitable for studies that require a closed system without external influences. The canonical (NVT) ensemble, which maintains a constant temperature by exchanging energy with a heat bath, employs both the Langevin thermostat[34] and Nose–Hoover integrators.[35,36] The Langevin thermostat implements the Brünger–Brooks–Karplus scheme for sampling the NVT ensemble. The current implementation of the Nose–Hoover thermostat uses only one bead. The multi-bead thermostat is currently being developed. We are also adding the BAOAB[37] and middle scheme[38] integrators for this ensemble.

apoCHARMM uses the SHAKE algorithm[39] for holonomic constraints related to hydrogen bonds at the two-, three-, and four-atom sites. For the more efficient non-iterative handling of the constraints related to TIP3P water molecules, apoCHARMM uses the SETTLE algorithm.[40] These methods are crucial for reducing the degrees of freedom in constrained systems, enabling larger time steps and thus speeding up the simulation without compromising accuracy.

As noted previously, apoCHARMM calculates the full virial tensor during each force calculation, allowing the implementation of isobaric-isothermal (NPT) ensemble simulations using the Langevin piston method.[23] This extended ensemble method introduces additional degrees of freedom associated with virtual pistons and is based on the original Anderson scheme.[41] The motion of the piston degrees of freedom is governed by the Langevin equation of motion. This second-order dampening motion removes the “ringing” artifact that is associated with the piston degrees of freedom in the MTK algorithm.[42]

The equations of motion for NPH dynamics are given by

$$\dot{\mathbf{r}}_\mathbf{i} = \frac{\mathbf{p_i}}{m_i} + \frac{1}{3}\frac{\dot{V}}{V}\mathbf{r_i},$$

$$\dot{\mathbf{p}}_\mathbf{i} = \mathbf{f_i} - \frac{1}{3}\frac{\dot{V}}{V}\mathbf{p_i},$$

$$\ddot{V} = \frac{1}{W}[P(t) - P_{\text{ext}}] - \gamma\dot{V} + R(t).$$

Here, $\mathbf{r_i}$ and $\mathbf{p_i}$ are atom i’s position and momentum, respectively, γ is the collision frequency, W is the piston mass, $R(t)$ is a random force drawn from a Gaussian distribution with mean value 0 and a variance of

$$\langle R(0)\cdot R(t)\rangle = \frac{2\gamma k_b T\delta(t)}{W},$$

where k_b is the Boltzmann constant. Furthermore, for the NPT ensemble, the temperature is controlled by coupling the Langevin piston dynamics with the Nose–Hoover equation of motion. Note that, using $\gamma = 0$ yields the MTK algorithm.

1. Supported crystal geometries

apoCHARMM supports several crystal geometries to accommodate different system symmetries and constraints on volume scaling. For cubic crystals, isotropic scaling is applied uniformly in all directions with a single degree of freedom, making it well-suited for systems requiring homogeneous expansion, such as bulk solvent or isotropic solids. Tetragonal systems, by contrast, introduce anisotropic scaling, coupling the X and Y dimensions to share one degree of freedom while allowing the Z dimension to vary independently. This configuration provides flexibility in the XY-plane, making it particularly useful for simulating structures such as lipid bilayers, where planar fluctuations are necessary. By contrast, orthorhombic crystal systems are handled with independent pistons for the X, Y, and Z dimensions, allowing each dimension to scale independently and the system to adapt dynamically in all three axes.

The versatility of these ensemble configurations extends further through options for specific surface and area constraints. For instance, apoCHARMM supports constant area (NPAT) and constant surface tension (NPγT) ensembles, which are indispensable in studying interfaces and membrane systems. Constant area simulations allow lipid bilayers or interfacial systems to maintain stable lateral dimensions while allowing other properties, such as pressure or tension, to fluctuate. The constant surface tension ensemble enables the accurate modeling of lipid bilayers under specific tension conditions, mimicking biological membranes under stress or surface tension.

2. Liquid/liquid interface ensembles

Several liquid/liquid interface ensembles can be simulated using the previously described crystal types made possible by the Langevin piston integrator. Some of these ensembles are the following:

- Constant NP_nAH_{pn}: The lengths of the box along the X- and Y-axes remain unchanged, giving a constant area, while the height of the box (Z-axis) is allowed to fluctuate. The masses of the piston variables corresponding to the length and the width of the box are set to infinity.

- Constant $NP_t h_z H_{pt}$: The height (h_z) of the simulation box is fixed, while the length (h_x) and width (h_y) are dynamic variables and allowed to fluctuate. The mass of the piston variable corresponding to the height is set to infinity.
- Constant $NV\gamma H_{v\gamma}$: The length (h_x) and width (h_y) of the simulation box are dynamic variables and the height (h_z) is allowed to vary such that volume (V) and surface tension (γ) remain constant.
- Constant $NP_n\gamma H_{p\gamma}$: All three lengths of the simulation box h_x, h_y, and h_z are allowed to vary, while the surface tension (γ) remains constant.

Each of these ensembles can be sampled in an adiabatic way by removing the thermostat or in a non-adiabatic way by coupling them with the Nose–Hoover thermostat (the default option). The Lagrangian, Hamiltonian, and equations of motion for these ensembles are available on the apoCHARMM website.

C. $P2_1$ periodic boundary conditions

$P2_1$ periodic boundary conditions have also been implemented in the apoCHARMM package.[30,31] The $P2_1$ boundary condition helps alleviate differential stress between lipid bilayer leaflets during MD simulations.[43] This stress arises from the inherent difficulty of accurately determining the number of lipids in each leaflet beforehand. The conventional periodic boundary condition, $P1$, typically involves replicating the unit cell through direct translations, that is, the only symmetry present in this PBC is the translational symmetry. Following the $P1$ condition, atoms leaving one side of the box are replaced by their counterparts entering from the opposite side, allowing the simulation to mimic bulk behavior. In contrast, the $P2_1$ periodic boundary condition is specifically tailored to enable lipid exchange between bilayer leaflets. Unlike the simple translational symmetry of $P1$, $P2_1$ introduces a half-screw symmetry. This means that the image of the simulation box is not merely translated but also rotated 180° around the screw axis during the translation along one of the axes. Within the CHARMM non-DOMDEC framework, this screw axis can be oriented in any direction, providing flexibility in its application. However, without loss of generality, the simulation box can be rotated such that the screw axis aligns with the X-axis. apoCHARMM implements the X-axis as the screw axis. A manuscript with further implementation and theoretical details of the $P2_1$ PBC in apoCHARMM is currently under preparation.

D. Free energy calculations

Several free energy difference methods have been implemented in apoCHARMM. Many of these methods can be represented as energy interpolation between two or more states.[44] To that end, apoCHARMM is specifically designed using the composite software design pattern to implement the methods under a unifying scheme (see Sec. III A Code Architecture for more details). In this scheme, forces and energies of the end states, after being calculated separately for each end state, are interpolated using the chosen flavor of interpolation scheme. For example, CHARMM's implementation of free energy perturbation in the PERT module samples an intermediary λ state by linear interpolation between the energies of the two end states. In apoCHARMM, each energy call is independent, and the composite object reweighs the forces (after the forces for all the states are calculated) to get the force to sample the phase space.

In the more recent Enveloping Distribution Sampling (EDS) method,[25–27] a reference state that envelops the important regions of the configurational phase space of all end states is sampled. The reference Hamiltonian is specified such that its partition function is the sum of the partition functions of the end states. The functional form of the reference Hamiltonian is parameterized by two sets of factors—a smoothness parameter and energy offsets for each end state:

$$V_R(\mathbf{r}) = -(\beta s)^{-1}\ \ln\left[e^{-\beta s(V_A(\mathbf{r})-E_A^R)} + e^{-\beta s(V_B(\mathbf{r})-E_B^R)}\right],$$

where s is a smoothness parameter, and E_A^R and E_B^R are the energy offsets for the states A and B, respectively. By sampling the reference state R, the free energy difference between states A and B can be calculated as

$$\Delta F_{BA} = \Delta F_{BR} - \Delta_{AR} = -\beta^{-1}\ \ln\frac{\langle e^{-\beta(E_B-E_R)}\rangle_R}{\langle e^{-\beta(E_A-E_R)}\rangle_R},$$

where $\langle \ldots \rangle_R$ denotes the ensemble average over the reference state.

The EDS scheme can also be used to perform constant pH simulation.[28,29] The apoCHARMM implementation gives an explicit solvent EDS-based scheme to perform constant pH simulations. The goal in this case is to calculate the free energy difference between the protonated (HA) and the deprotonated (A^-) states of the protein. The pH-dependent free energy difference between the two states is given by

$$\Delta G_{protein} = \Delta G^{MM}_{protein} - \Delta G^{MM}_{model} + k_B T\ \ln\ 10(pH - pK_{a,model}),$$

where the ΔG^{MM}_{model} is first calculated for a model system, typically the amino acid of interest. This scheme assumes that the non-molecular mechanics (MM) contribution to the free energy difference is the same for the model and the system of interest and hence can be canceled out.

The multiple Hamiltonian scheme is used in the implementation of other free energy difference methods other free energy difference methods, such as Serial Atom Insertion (SAI).[45] In this scheme, the disappearing atoms are switched off one atom at a time until a common core between the two end states is reached. The currently implemented heuristic is taken from Ref. 46.

In addition, to avoid the endpoint catastrophe while switching off the van der Waals terms, the soft-core formulation of the van der Waals interactions[47] to calculate λ-specific energy is available. Although it is slightly slower than the van der Waals formulation, the double exponential method[48] has been implemented as well and provides a base version of the soft-core potential.

The parameter interpolation method is implemented through Python modules provided with the apoCHARMM package. In this scheme, rather than interpolating the energies and forces, separate PSFs and PRMs corresponding to the force field parameters for the intermediate states are generated. Each of the intermediate states can be sampled in an independent simulation. In addition, the intermediate states can also be sampled through a child object of the composite scheme where each state is sampled sequentially, but the energies in all of the states are evaluated on the device with high frequency. Since coordinates do not need to be stored for post-processing and the collected energies of all states are stored in device memory, this method enables excellent exploration of free energy differences at a high frequency.

A single-topology scheme for relative free energy calculations has been implemented, and a dual-topology scheme is currently under development. In these schemes, the maximum common substructure is identified and the differing atoms between the two states are handled as dummy atoms. Non-bonded exclusions ensure that the dummy atoms only have bonded interactions. Another manuscript containing the details of the parameter interpolation relative free energy difference calculation is under preparation.

E. Replica exchange

Replica exchange, also known as parallel tempering, is a computational technique used in MD simulations to enhance sampling efficiency, especially for complex systems with rugged energy landscapes. In replica exchange, multiple replicas of the system are simulated concurrently at different temperatures, allowing them to explore a variety of configurations. Periodically, neighboring replicas exchange configurations based on a probability that preserves thermodynamic properties, ensuring accurate sampling across the energy landscape. This approach enables low-temperature replicas to escape local energy minima by exchanging with high-temperature replicas, making it particularly useful for systems with slow dynamics, such as protein folding and biomolecular conformational sampling.

In apoCHARMM, we have implemented both multi-GPU as well as single-GPU versions of replica exchange sampling. A salient feature of the replica exchange implementation in apoCHARMM is that we can perform (up to) 48 replicas on the same GPU. This functionality is particularly useful in cases where the individual system size of each replica is not large enough to saturate the streaming multiprocessors (SMs) on the GPU. This feature will be even more important for future GPUs as the number of SMs increases with every generation.

F. Minimization

A number of minimizers that have been implemented in apoCHARMM. The steepest descent (SD) algorithm uses a first-order scheme with an adaptive step size. There are two second-order minimizers that have been implemented, namely the Adaptive Basis Newton–Raphson (ABNR) and the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithms. To perform the minimization under holonomic constraints, the geometry is first constrained and then the gradient along the constraints is projected out.

III. IMPLEMENTATION

A. Code architecture

apoCHARMM originates from an earlier GitHub package developed by Antti-Pekka Hynninen[49] and has been re-engineered in CUDA and modern C++ to fully exploit NVIDIA GPU architectures. It offers a pybind11-based Python interface, enabling a user-friendly, “pythonic” interaction for end users. Built following Test Driven Development (TDD) principles, apoCHARMM’s codebase features Catch2-based unit tests that ensure extensive code coverage and robust functionality. Designed as a GPU-exclusive platform, apoCHARMM handles all molecular dynamics calculations—energy, force, restraints, constraints, and integration—directly on the GPU, reducing host-GPU memory transfers to only those needed for logging and trajectory storage.

The code architecture of apoCHARMM reflects our attempt at modularity and modern design principles. A schematic view of the architecture is presented in the Fig. 1. At its core, the CharmmContext object acts as the central mediator, facilitating communication between various components. The ForceManager governs the functional form of energy and force calculations. In addition, two example subscribers are depicted—one responsible for reporting trajectory output in DCD format and another for generating a state description—both interfacing with the integrator. The Integrator class has been derived to create specific schemes of thermostats, barostats, and other integrators.

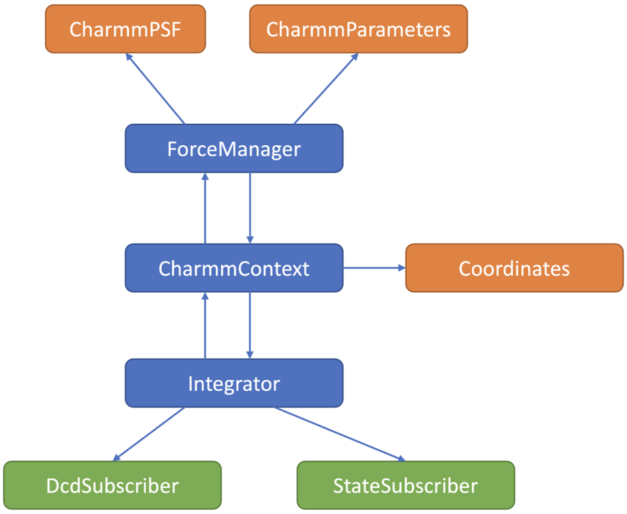

FIG. 1. A schematic representation of the code architecture. At its core, the CharmmContext object acts as the central mediator, facilitating communication between various components. The ForceManager governs the functional form of energy and force calculations. In addition, two example subscribers are depicted—one responsible for reporting trajectory output in DCD format and another for generating a state description—both interfacing with the integrator. Orange reflects objects holding external information provided as inputs, blue are the objects that implement the molecular dynamics algorithms. Green-colored objects are the reporter objects, which are the outputs of the MD simulation.

Modern software designs follow a number of structural and behavioral design patterns. We have architected apoCHARMM to follow the ones presented in Fig. 2. The mediator design pattern is implemented to reduce dependencies among components by centralizing communication through a single mediator object. CharmmContext uses this design pattern, which ensures that the communication between two objects (like the force calculation during integration or minimization) is interfaced through the CharmmContext, such that the interface remains unchanged. The composite design pattern allows a class to be a child of itself and thus form a tree-like structure of objects. ForceManager in apoCHARMM follows this pattern, where ForceManagers of multiple different Hamiltonians form the leaves of this composite pattern. In addition, a publisher-subscriber pattern supports modularity by allowing loggers and integrators to interact flexibly, making it possible to pair different loggers with various integrators. This modular approach enhances the adaptability and extensibility of apoCHARMM, making it an effective, high-performance tool.

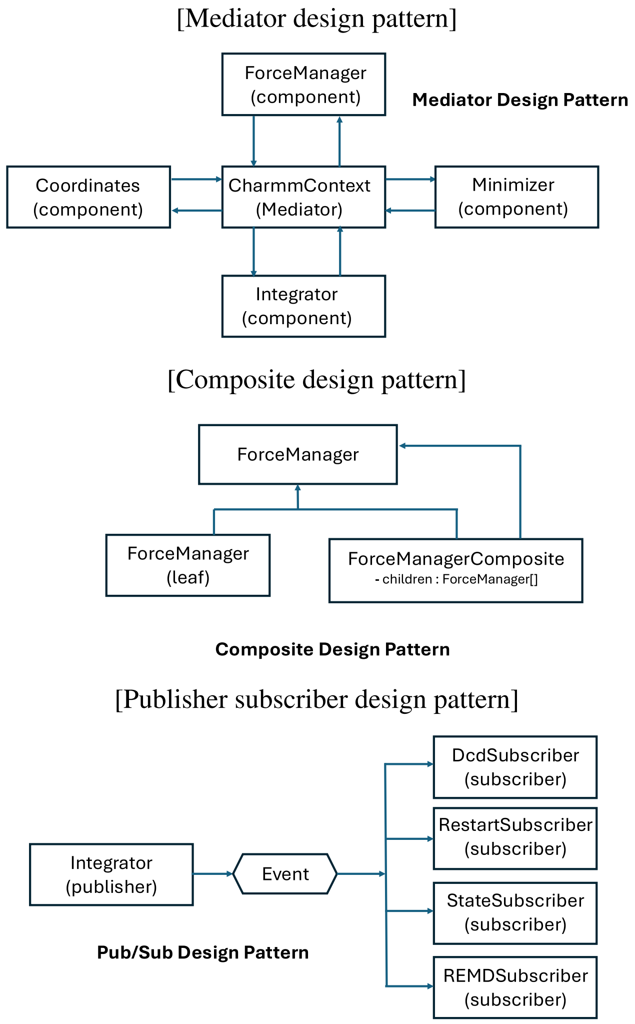

FIG. 2. Design patterns in apoCHARMM.

B. Neighbor list preparation

The most computationally expensive component of the integration of each dynamic step is evaluating the short-range nonbonded forces (electrostatic and van der Waals interactions). Several optimizations were implemented by MD simulation packages to fine-tune this calculation. One of the most important schemes is that of neighbor list preparation. Rather than calculating the pairwise interactions between all atoms, a neighbor list data structure keeps track of the interacting pairs. This list is periodically updated (usually every 15–20 steps), since atoms would have moved out of the region at the time of preparation of the list. A buffer region is also included so that the frequency of neighbor list updating can be minimized. The cost of the neighbor list preparation is amortized over the number of steps where it is used.

Unlike central processing units (CPUs), where accessing scattered memory locations is relatively manageable due to the larger cache sizes, GPUs are optimized for coalesced memory access. However, constructing and using neighbor lists involves fetching data for atom pairs that are often stored at non-contiguous locations in memory. Since the sequence of atom indices does not necessarily follow their spatial location, this results in uncoalesced memory fetches. Consequently, the GPU's ability to efficiently load data in a streamlined, vectorized manner is severely hindered. This irregular memory access pattern leads to suboptimal utilization of memory bandwidth, increasing latency and reducing overall computational efficiency. Hence, we first sort the atoms in spatially sequential format.

Atoms are first divided into cells containing 32 atoms each. To do this, the maximum and minimum coordinates of the atoms in the simulation box are calculated using an efficient parallel maximum and minimum reduction algorithm. Atoms are then divided into cells where each cell has 32 atoms. This is done by evenly dividing the X and Y dimensions into evenly sized lengths and creating the boundaries at each column such that the cuboidal cell has exactly 32 atoms. This is implemented using a parallel exclusive prefix sum.

Atoms are sorted in two stages. First, we rearrange them according to the column that they lie in. Next, we use a shared memory-based bitonic sort to sort the atoms along each column. Now that we have spatially sorted coordinates and created cells containing 32 atoms, the neighbor list can be built. Each cell is handled by a CUDA warp (group of 32 threads). The neighbor list construction kernel is implemented using CUDA and designed to efficiently construct neighbor lists for particle-based simulations by leveraging a cell-based spatial decomposition scheme. The kernel operates under the self-interaction model, where each cell interacts only with itself and its designated neighboring cells within a predefined cutoff radius. This computational approach ensures that all necessary neighbor pairs are identified while minimizing redundant distance calculations and memory accesses.

Each warp is assigned a single simulation cell, ensuring efficient load balancing across the computational grid. The global index of the assigned cell is determined using a combination of thread and block indices, after which the kernel retrieves its spatial coordinates and corresponding zone index from global memory. To optimize memory access patterns and reduce global memory transactions, the bounding box of the assigned cell is loaded into shared memory, allowing for rapid access by all threads within the warp. This data locality enhancement significantly improves computational efficiency, as repeated accesses to global memory are avoided.

The kernel iterates over neighboring cells in three spatial dimensions (including the central cell itself), checking for potential interactions. To minimize unnecessary pairwise distance calculations, a bounding box overlap check is performed first, using thread-cooperative shared memory operations. This initial filtering step eliminates distant cells that cannot contribute to the neighbor list, reducing the number of pairwise evaluations required in subsequent steps. The remaining candidate neighbor cells are stored in a shared memory buffer, dynamically allocated based on the problem size and GPU architecture constraints.

For each valid neighboring cell, a thread-cooperative loop over particle pairs is executed. Each thread within the warp is responsible for computing the squared distance between a single particle from the current cell and a particle from the neighboring cell. The squared distance is computed efficiently using register-level operations, and interactions exceeding the cutoff radius are immediately discarded. To ensure coalesced memory access, the neighbor list entries are first temporarily stored in shared memory before being committed to global memory in batched writes, minimizing memory transaction overhead.

The kernel further exploits warp-level parallelism using shuffle instructions and shared memory-based reductions to optimize intra-warp communication and load balancing. These techniques help maximize hardware occupancy and minimize warp divergence caused by conditional branching in pairwise computations. The compact storage format of neighbor lists allows for efficient retrieval in subsequent force evaluation steps, reducing the overhead of repeated memory accesses.

Next, the exclusion lists are built. To efficiently construct exclusion lists for MD simulations, we developed a CUDA kernel that processes atomic interactions within spatially partitioned cells. The kernel operates on a per-warp basis, with each warp handling an individual spatial cell and iterating over its constituent atoms. Shared memory optimizations are employed for architectures below compute capability 3.0 to store shuffle memory buffers and atomic coordinate data, ensuring efficient inter-thread communication. Each thread computes exclusions by first determining the local-to-global and global-to-local atom mappings, followed by retrieving atom-specific exclusion data from a precomputed exclusion list. A hierarchical approach is used to manage exclusions: the kernel first assigns exclusion space in global memory and loads per-atom exclusion data. Using warp-wide prefix sum operations, each thread calculates the total number of excluded atoms. Subsequently, exclusion lists are populated by iterating over neighbor tiles, where atoms are loaded into either registers or shared memory, depending on the compute capability. For efficient pairwise distance calculations, atom coordinates are translated according to periodic boundary conditions, incorporating symmetry operations for specific crystal symmetries when enabled. The kernel employs shuffle-based and shared memory-based reductions to determine the minimum and maximum excluded atom indices per warp, which are then used to construct exclusion masks at the tile level. To improve memory locality and reduce branching divergence, the kernel employs warp-wide bitwise operations to encode exclusion masks efficiently. In addition, a secondary exclusion check is performed for short-range interactions within a predefined cutoff distance, ensuring that atoms satisfying exclusion criteria are appropriately flagged and recorded. The exclusion lists are finally structured into buckets for sorting, leveraging atomic operations to increment position counters in global memory.

C. Force calculation

In apoCHARMM, each type of force calculation is executed on a dedicated CUDA stream, enabling parallelization of idle SMs that are available on the GPU. By offloading different calculations to separate streams, apoCHARMM maximizes GPU utilization, taking advantage of available hardware resources to maintain continuous computational flow. The calculations for bonded interactions—such as bonds, angles, Urey–Bradley, proper and improper dihedrals, and CMAP terms—are among the least computationally intensive and thus suited for efficient parallel handling. Separate kernels are assigned to each bonded interaction type, and these kernels are organized within CUDA graphs to minimize the overhead associated with launching multiple kernels. Similarly, restraints are processed within their own stream, ensuring that constraints are efficiently managed without interrupting other force calculations.

Non-bonded interactions in apoCHARMM are calculated using an Ewald-based method by default, which divides the workload between direct and reciprocal space streams. This provides control over the execution order and avoids the need for global synchronization after every kernel launch. This division allows the engine to split these calculations, with the direct space stream maintaining an up-to-date neighbor list for atom interactions. To keep the neighbor list optimized, a narrow buffer region extends beyond the cutoff radius, requiring the list to be updated only every 15–20 simulation steps. For efficiency, atoms in the simulation cell are organized into spatial cells, each containing 32 atoms. A precomputed list of potential interaction cells within the cutoff range is stored in the neighbor list, allowing each CUDA warp to manage interactions between a specific pair of cells, known as a “tile.” As the warp iterates over these tiles, it calculates both electrostatic and van der Waals interactions for atoms within the cutoff range, excluding designated non-interacting pairs. Both force-switching and potential-switching variants of the interaction are supported, and an isotropic periodic sum (IPS) formulation is also available for calculating the full non-bonded interaction using only short-range calculations.[50]

The long-range portion of the non-bonded calculations, handled in the reciprocal space stream, follows the Particle Mesh Ewald (PME) method,[51] which involves five computational steps. First, atomic charges are mapped onto a three-dimensional grid using B-spline interpolation; either the grid dimensions can be manually specified by the user, or they can be automatically determined based on a desired error threshold. Once the charges are positioned on the grid, a Fourier transform is applied to convert the data into reciprocal space using NVIDIA's cuFFT library. In reciprocal space, convolution with a precomputed kernel is performed, from which the energy and virial contributions are computed. An inverse Fourier transform is then applied to the resulting data, and forces are calculated by probing the real-space grid using derivative splines, providing the precise forces needed for the next simulation steps.

Forces are calculated in single precision but accumulated in fixed precision.[52] This avoids the problem associated with floating point addition being non-associative. This scheme scales up the FP32 float by 2^{40}, accumulates and stores it as a 64-bit integer (long long int in C++), and converts it back to floating-point representation and scales down by 2^{-40}.

IV. USAGE

A fundamental demonstration of the apoCHARMM MD package is presented in Listing 1. End users are encouraged to utilize the Python front end, which leverages a pybind11-based Python interface to the backend C++/CUDA codebase. This design ensures that apoCHARMM is user-friendly and seamlessly integrates with the Python ecosystem of molecular dynamics tools, enabling researchers to streamline their workflows while benefiting from Python's extensive scientific libraries.

The input data for apoCHARMM simulations include the CHARMM force field parameter files (PRMs) and the protein structure file (PSF). These files can be conveniently generated using established tools such as CHARMM-GUI[53] or CHARMM itself.[22] At present, the package supports only the CHARMM-formatted parameter files; however, development is underway to include compatibility with Amber format files. Future iterations of apoCHARMM aim to expand this support to additional force field formats, including OpenFF and OPLS, thereby enhancing its versatility and broadening its appeal to a wider range of researchers.

Simulation outputs in apoCHARMM are managed through a publisher-subscriber (pub/sub) design pattern, a robust approach for enabling asynchronous communication between the simulation engine and its outputs. The package currently implements several subscribers, which include functionalities such as

- Restart File Subscriber: Saves checkpoint files for resuming simulations.
- Trajectory Subscribers: Supports DCD and NetCDF formats for trajectory data.
- Debugging Outputs: Generates XYZ format files primarily for debugging purposes.
- State Subscriber: Captures and stores key simulation metrics, including potential energy, kinetic energy, temperature, and surface tension.
- CHARMM-Formatted Subscriber: Outputs data in CHARMM-compatible formats.
- Enveloping Distribution Sampling (EDS) Subscriber: Facilitates EDS calculations.
- Replica Exchange (REMD) Subscriber: Enables REMD simulations.

LISTING 1. apoCHARMM usage example for an NVT followed by an NPT run.

```python
from charmm import apocharmm as ch
import argparse

def simulation(args):
    prmlist = [’par_all36_lipid.prm’,
                    ’toppar_all36_lipid_bacterial.str’, ’toppar_water_ions.str’]

    prm = ch.CharmmParameters(prmlist)
    psf = ch.CharmmPSF("system.psf")
    # crd = ch.CharmmCrd("equil.crd")
    crd = ch.CharmmCrd("equil.cor")

    fm = ch.ForceManager(psf, prm)
    fm.setBoxDimensions([147.09442, 147.09442, 106.35873])
    fm.setCutoff(12.0)
    fm.setCtonnb(8.0)
    fm.setCtofnb(10.0)

    ctx = ch.CharmmContext(fm)

    ctx.setCoordinates(crd)
    ctx.assignVelocitiesAtTemperature(300)

    # An initial NVT simulation
    langevinThermostat = ch.LangevinThermostatIntegrator(0.002)
    langevinThermostat.setFriction(5.0)
    langevinThermostat.setBathTemperature(298.17)
    langevinThermostat.setSimulationContext(ctx)
    restartsubscriber = ch.RestartSubscriber("out/nvt.res", 10000)
    langevinThermostat.subscribe([restartsubscriber])
    numSteps = int(1e5)
    langevinThermostat.propagate(numSteps)
    print("NVT done")

    # An additional short NPT simulation to get the right area
    langevinPiston = ch.LangevinPistonIntegrator(0.002)
    langevinPiston.setCrystalType(ch.CRYSTAL.TETRAGONAL)
    #langevinPiston.setPistonMass(500.0)
    langevinPiston.setPistonFriction(12.0)
    langevinPiston.setBathTemperature(298.17)
    langevinPiston.setSimulationContext(ctx)
    subscriber = ch.DcdSubscriber("out/npt.dcd", 1000)
    restartsubscriber = ch.RestartSubscriber("out/npt.res", 10000)
    langevinPiston.subscribe([subscriber, restartsubscriber])
    nptSteps = int(1e5)
    langevinPiston.propagate(nptSteps)
    print("Short NPT done")

    # A production NPT simulation
    langevinPiston = ch.LangevinPistonIntegrator(0.002)
    langevinPiston.setCrystalType(ch.CRYSTAL.TETRAGONAL)
    langevinPiston.setPistonMass([0.0, 500.0])
    langevinPiston.setPistonFriction(12.0)
    langevinPiston.setBathTemperature(298.17)
    langevinPiston.setSimulationContext(ctx)
    subscriber = ch.DcdSubscriber("out/npat.dcd", 10000)
    restartsubscriber = ch.RestartSubscriber("out/npat.res", 50000)
    langevinPiston.subscribe([subscriber, restartsubscriber])
    nptSteps = int(1e7)
    langevinPiston.setDebugPrintFrequency(1000)
    langevinPiston.propagate(nptSteps)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description=’Pore simulation’)
    args = parser.parse_args()
    simulation(args)
```

LISTING 2. Example of using the EDSForceManager.

```
...
    # Import the parameters anf the system's topology (prm and psf)
    psf0 = ch.CharmmPSF("glu_depr.psf")
    psf1 = ch.CharmmPSF("glu_prot.psf")

    prm = ac.CharmmParameters(
        ["toppar_water_ions.str", "par_all36m_prot.prm"]
    )

    # Set up the ForceManager that will drive the simulation.
    # Create the ForceManager object using the psf and prm
    fm0 = ch.ForceManager(psf0, prm)
    fm1 = ch.ForceManager(psf1, prm)

    fmEDS = ch.EDSForceManager(fm0, fm1)

    # Setup box size, FFT options, cutoffs...
    fmEDS.setBoxDimensions([30.0, 30.0, 30.0])
    fmEDS.setFFTGrid(30, 30, 30)
    fmEDS.setCtonnb(9.0)
    fmEDS.setCtofnb(10.0)
    fmEDS.setCutoff(12.0)
    # Finally, initialize the ForceManager object !
    fmEDS.initialize()

    s = 0.003
    eOffset0 = 0.0
    eOffset1 = 44.0

    fmEDS.setSValue(s)
    fmEDS.setEnergyOffsets([eOffset0, eOffset1])

    # The simulation state will be handled by a CharmmContext object, created from the ForceManager.
    ctx = ch.CharmmContext(fmEDS)
...
```

V. TESTING AND VALIDATION

Here we provide a working example for setting up a replica exchange simulation. More examples of the usage can be found in the tutorials, in the GitHub repository.[54]

We compared the performance of apoCHARMM on a suite of biomolecular systems over a set of recent Nvidia GPU architectures. The results are shown in Fig. 3 and Table II. The systems tested for performance were ApoAI (92224 atoms), DMPG (291168 atoms), and STMV (1066628) on Volta, Ampere, and Hopper architectures. Pressure conservation plots for a waterbox using cubic, tetragonal, and orthorhombic boxes are shown in Fig. 4. The target pressure for these runs was set to 1 atm. All tests use a 9 Å cutoff for short range, 0.34 as the kappa factor for Ewald calculation. The timestep for these runs is 2 fs. apoCHARMM is being actively optimized; therefore, we recommend users check the website for the latest performance numbers and/or test the performance on their machines to determine the time allocation estimation for their runs. More examples of usage, tutorials, and benchmarks can be found on the project webpage.

TABLE II. apoCHARMM performance (ns/day) on different GPUs. System sizes are mentioned in brackets. Architectures in these tests are Volta V100, Ampere A100, and Hopper H100.

System	V100	A100	H100
APOI (92224)	120.5	172.5	246.6
DMPG (291168)	40.7	66.2	114.7
STMV (1066628)	9.8	18.7	32.5

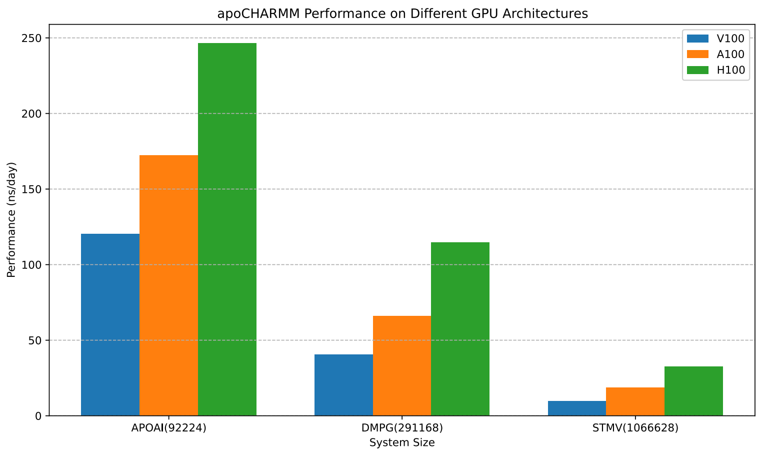

FIG. 3. Performance of apoCHARMM on three different generations of GPU architectures (Volta V100, Ampere A100, and Hopper H100) for three molecular systems (APOAI, DMPG, and STMV).

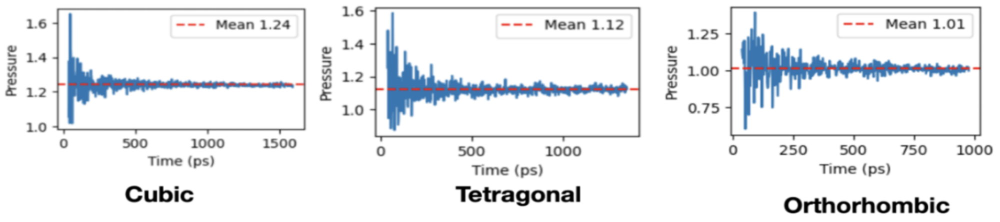

FIG. 4. Pressure conservation plot for cubic, tetragonal, and orthorhombic crystal types for a waterbox using the Langevin piston method.

VI. CONCLUSIONS

In summary, apoCHARMM represents a significant advancement in MD simulation technology by using GPU acceleration to deliver high-performance, flexible simulations. With the ability to compute the full virial tensor, apoCHARMM enables accurate simulations across a range of ensembles, including NPT, NVT, and others, supporting diverse applications in biophysical research and drug discovery. Its design specifically addresses the computational challenges associated with free energy calculations by combining speed and precision. Through our benchmarks and test cases, we demonstrated that apoCHARMM offers substantial improvements in simulation efficiency and scalability compared to conventional CPU-based MD engines, while maintaining the rigorous accuracy required for detailed thermodynamic and structural analyses.

Looking forward, apoCHARMM has strong potential for broad adoption across the molecular modeling community, particularly in fields that require extensive sampling and high-resolution free energy landscapes. Future work will continue to expand apoCHARMM's capabilities, including optimizing additional advanced sampling methods and integrating hybrid CPU–GPU functionalities for even greater performance flexibility. By providing a robust, versatile tool that maximizes the computational power of GPUs, apoCHARMM paves the way for more comprehensive and accessible simulations of complex biological systems. We anticipate that apoCHARMM will facilitate new discoveries and advancements in computational biology, bringing high-performance MD simulations within reach for a wider range of researchers and applications.

We are currently working on several optimizations. We are also working on changing the tile size from 32 atoms to smaller 16 and 8 atoms to improve the performance. The reordering of the atoms at the sorting stage of neighbor list preparation is currently being done at the column-level granularity. This is suboptimal since, even after reordering, consecutive atoms may lie anywhere in a cell. We will be moving this to instead perform fractal sorting. The package is open source and available under the BSD 3-Clause license on GitHub at apocCHARMM.[54]

ACKNOWLEDGMENTS

The authors declare no conflict of interest. F.A. worked on it during this time at NIH. This work was performed on the LoBoS supercomputing cluster and the NIH high-performance computing cluster Biowulf. This research was supported by the Intramural Research Program of the NIH, NHLBI (Grant No. ZIA HL001050). J.E.G. was also partially supported by the National Institute of General Medical Sciences of the NIH under Award No. R01GM127723.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Samarjeet Prasad: Conceptualization (lead); Data curation (lead); Formal analysis (lead); Investigation (lead); Methodology (lead); Project administration (equal); Resources (equal); Software (equal); Supervision (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Felix Aviat**: Software (supporting); Writing – review & editing (supporting). **James E. Gonzales**: Software (supporting); Writing – review & editing (equal). **Bernard R. Brooks**: Funding acquisition (lead); Project administration (lead); Supervision (lead); Writing – review & editing (equal).

DATA AVAILABILITY

The package is open source and available under the BSD 3-Clause license on GitHub at https://github.com/samarjeet/apocharmm.[54]

REFERENCES

[1] D. E. Shaw, P. J. Adams, A. Azaria, J. A. Bank, B. Batson, A. Bell, M. Bergdorf, J. Bhatt, J. A. Butts, T. Correia *et al.*, "Anton 3: Twenty microseconds of molecular dynamics simulation before lunch," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Association for Computing Machinery, 2021), pp. 1–11.

[2] D. E. Shaw, J. P. Grossman, J. A. Bank, B. Batson, J. A. Butts, J. C. Chao, M. M. Deneroff, R. O. Dror, A. Even, C. H. Fenton *et al.*, "Anton 2: Raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer," in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis* (IEEE, 2014), pp. 41–53.

[3] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao *et al.*, "Anton, a special-purpose machine for molecular dynamics simulation," Commun. ACM **51**, 91–97 (2008).

[4] P. Eastman and V. Pande, "OpenMM: A hardware-independent framework for molecular simulations," Comput. Sci. Eng. **12**, 34–39 (2010).

[5] P. Eastman, M. S. Friedrichs, J. D. Chodera, R. J. Radmer, C. M. Bruns, J. P. Ku, K. A. Beauchamp, T. J. Lane, L.-P. Wang, D. Shukla *et al.*, "OpenMM

4: A reusable, extensible, hardware independent library for high performance molecular simulation," J. Chem. Theory Comput. **9**, 461–469 (2013).

[6] P. Eastman, J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern *et al.*, "OpenMM 7: Rapid development of high performance algorithms for molecular dynamics," PLoS Comput. Biol. **13**, e1005659 (2017).

[7] P. Eastman, R. Galvelis, R. P. Peláez, C. R. A. Abreu, S. E. Farr, E. Gallicchio, A. Gorenko, M. M. Henry, F. Hu, J. Huang *et al.*, "OpenMM 8: Molecular dynamics simulation with machine learning potentials," J. Phys. Chem. B **128**, 109–116 (2023).

[8] B. Kohnke, C. Kutzner, and H. Grubmüller, "A GPU-accelerated fast multipole method for GROMACS: Performance and accuracy," J. Chem. Theory Comput. **16**, 6938–6949 (2020).

[9] S. Páll, A. Zhmurov, P. Bauer, M. Abraham, M. Lundborg, A. Gray, B. Hess, and E. Lindahl, "Heterogeneous parallelization and acceleration of molecular dynamics simulations in GROMACS," J. Chem. Phys. **153**, 134110 (2020).

[10] A. W. Götz, M. J. Williamson, D. Xu, D. Poole, S. Le Grand, and R. C. Walker, "Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. Generalized born," J. Chem. Theory Comput. **8**, 1542–1555 (2012).

[11] R. Salomon-Ferrer, A. W. Götz, D. Poole, S. Le Grand, and R. C. Walker, "Routine microsecond molecular dynamics simulations with AMBER on GPUs. 2. Explicit solvent particle mesh Ewald," J. Chem. Theory Comput. **9**, 3878–3888 (2013).

[12] T.-S. Lee, Y. Hu, B. Sherborne, Z. Guo, and D. M. York, "Toward fast and accurate binding affinity prediction with pmemdGTI: An efficient implementation of GPU-accelerated thermodynamic integration," J. Chem. Theory Comput. **13**, 3077–3084 (2017).

[13] T.-S. Lee, D. S. Cerutti, D. Mermelstein, C. Lin, S. LeGrand, T. J. Giese, A. Roitberg, D. A. Case, R. C. Walker, and D. M. York, "GPU-accelerated molecular dynamics and free energy methods in Amber18: Performance enhancements and new features," J. Chem. Inf. Model. **58**, 2043–2050 (2018).

[14] T. J. Giese and D. M. York, "A GPU-accelerated parameter interpolation thermodynamic integration free energy method," J. Chem. Theory Comput. **14**, 1564–1582 (2018).

[15] J. Jung, A. Naurse, C. Kobayashi, and Y. Sugita, "Graphics processing unit acceleration and parallelization of genesis for large-scale molecular dynamics simulations," J. Chem. Theory Comput. **12**, 4947–4958 (2016).

[16] C. Kobayashi, J. Jung, Y. Matsunaga, T. Mori, T. Ando, K. Tamura, M. Kamiya, and Y. Sugita, "GENESIS 1.1: A hybrid-parallel molecular dynamics simulator with enhanced sampling algorithms on multiple computational platforms," J. Comput. Chem. **38**, 2193 (2017).

[17] J. Jung, K. Yagi, C. Tan, H. Oshima, T. Mori, I. Yu, Y. Matsunaga, C. Kobayashi, S. Ito, D. Ugarte La Torre, and Y. Sugita, "GENESIS 2.1: High-performance molecular dynamics software for enhanced sampling and free-energy calculations for atomistic, coarse-grained, and quantum mechanics/molecular mechanics models," J. Phys. Chem. B **128**, 6028 (2024).

[18] K. Nguyen-Cong, J. T. Willman, S. G. Moore, A. B. Belonoshko, R. Gayatri, E. Weinberg, M. A. Wood, A. P. Thompson, and I. I. Oleynik, "Billion atom molecular dynamics simulations of carbon at extreme conditions and experimental time and length scales," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Association for Computing Machinery, 2021), pp. 1–12.

[19] T. Yoshizawa, K. Uchibori, M. Araki, S. Matsumoto, B. Ma, R. Kanada, Y. Seto, T. Oh-Hara, S. Koike, R. Ariyasu *et al.*, "Microsecond-timescale MD simulation of EGFR minor mutation predicts the structural flexibility of EGFR kinase core that reflects EGFR inhibitor sensitivity," npj Precis. Oncol. **5**, 32 (2021).

[20] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, "CHARMM: A program for macromolecular energy, minimization, and dynamics calculations," J. Comput. Chem. **4**, 187–217 (1983).

[21] B. R. Brooks, C. L. Brooks III, A. D. Mackerell, Jr., L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch *et al.*, "CHARMM: The biomolecular simulation program," J. Comput. Chem. **30**, 1545–1614 (2009).

[22] W. Hwang, S. L. Austin, A. Blondel, E. D. Boittier, S. Boresch, M. Buck, J. Buckner, A. Caflisch, H.-T. Chang, X. Cheng *et al.*, "CHARMM at 45: Enhancements in accessibility, functionality, and speed," J. Phys. Chem. B **128**, 9976–10042 (2024).

[23] S. E. Feller, Y. Zhang, R. W. Pastor, and B. R. Brooks, "Constant pressure molecular dynamics simulation: The Langevin piston method," J. Chem. Phys. **103**, 4613–4621 (1995).

[24] M. R. Shirts and J. D. Chodera, "Statistically optimal analysis of samples from multiple equilibrium states," J. Chem. Phys. **129**, 124105 (2008).

[25] C. D. Christ and W. F. van Gunsteren, "Enveloping distribution sampling: A method to calculate free energy differences from a single simulation," J. Chem. Phys. **126**, 184110 (2007).

[26] C. D. Christ and W. F. Van Gunsteren, "Comparison of three enveloping distribution sampling Hamiltonians for the estimation of multiple free energy differences from a single simulation," J. Comput. Chem. **30**, 1664–1679 (2009).

[27] C. D. Christ and W. F. van Gunsteren, "Multiple free energies from a single simulation: Extending enveloping distribution sampling to nonoverlapping phase-space distributions," J. Chem. Phys. **128**, 174112 (2008).

[28] J. Lee, B. T. Miller, A. Damjanović, and B. R. Brooks, "Constant pH molecular dynamics in explicit solvent with enveloping distribution sampling and Hamiltonian exchange," J. Chem. Theory Comput. **10**, 2738–2750 (2014).

[29] J. Lee, B. T. Miller, A. Damjanović, and B. R. Brooks, "Enhancing constant-pH simulation in explicit solvent with a two-dimensional replica exchange method," J. Chem. Theory Comput. **11**, 2560–2574 (2015).

[30] E. A. Dolan, R. M. Venable, R. W. Pastor, and B. R. Brooks, "Simulations of membranes and other interfacial systems using $P2_1$ and Pc periodic boundary conditions," Biophys. J. **82**, 2317–2325 (2002).

[31] S. Prasad, A. C. Simmonett, R. Meana-Pañeda, and B. R. Brooks, "The extended eighth-shell method for periodic boundary conditions with rotational symmetry," J. Comput. Chem. **42**, 1373–1383 (2021).

[32] S. Park, A. Rice, W. Im, and R. W. Pastor, "Spontaneous curvature generation by peptides in asymmetric bilayers," J. Comput. Chem. **45**, 512 (2024).

[33] M. J. Louwerse and E. J. Baerends, "Calculation of pressure in case of periodic boundary conditions," Chem. Phys. Lett. **421**, 138–141 (2006).

[34] A. Brünger, C. L. Brooks III, and M. Karplus, "Stochastic boundary conditions for molecular dynamics simulations of ST2 water," Chem. Phys. Lett. **105**, 495–500 (1984).

[35] S. Nosé, "A unified formulation of the constant temperature molecular dynamics methods," J. Chem. Phys. **81**, 511–519 (1984).

[36] W. G. Hoover, "Canonical dynamics: Equilibrium phase-space distributions," Phys. Rev. A **31**, 1695 (1985).

[37] B. Leimkuhler and C. Matthews, "Robust and efficient configurational molecular sampling via Langevin dynamics," J. Chem. Phys. **138**, 174102 (2013).

[38] Z. Zhang, X. Liu, K. Yan, M. E. Tuckerman, and J. Liu, "Unified efficient thermostat scheme for the canonical ensemble with holonomic or isokinetic constraints via molecular dynamics," J. Phys. Chem. A **123**, 6056–6079 (2019).

[39] J.-P. Ryckaert, G. Ciccotti, and H. J. C. Berendsen, "Numerical integration of the cartesian equations of motion of a system with constraints: Molecular dynamics of *n*-alkanes," J. Comput. Phys. **23**, 327–341 (1977).

[40] S. Miyamoto and P. A. Kollman, "Settle: An analytical version of the SHAKE and RATTLE algorithm for rigid water models," J. Comput. Chem. **13**, 952–962 (1992).

[41] H. C. Andersen, "Molecular dynamics simulations at constant pressure and/or temperature," J. Chem. Phys. **72**, 2384–2393 (1980).

[42] G. J. Martyna, M. E. Tuckerman, D. J. Tobias, and M. L. Klein, "Explicit reversible integrators for extended systems dynamics," Mol. Phys. **87**, 1117–1157 (1996).

[43] S. Park, W. Im, and R. W. Pastor, "Developing initial conditions for simulations of asymmetric membranes: A practical recommendation," Biophys. J. **120**, 5041–5059 (2021).

[44] A. S. J. S. Mey, B. K. Allen, H. E. Bruce Macdonald, J. D. Chodera, D. F. Hahn, M. Kuhn, J. Michel, D. L. Mobley, L. N. Naden, S. Prasad *et al.*, "Best practices for alchemical free energy calculations [article v1.0]," Living J. Comput. Mol. Sci. **2**, 18378 (2020).

[45] S. Boresch and S. Bruckner, "Avoiding the van der Waals endpoint problem using serial atomic insertion," J. Comput. Chem. **32**, 2449–2458 (2011).

[46] M. Wieder, M. Fleck, B. Braunsfeld, and S. Boresch, "Alchemical free energy simulations without speed limits. A generic framework to calculate free energy differences independent of the underlying molecular dynamics program," J. Comput. Chem. **43**, 1151–1160 (2022).

[47] M. Zacharias, T. P. Straatsma, and J. A. McCammon, "Separation-shifted scaling, a new scaling method for Lennard-Jones interactions in thermodynamic integration," J. Chem. Phys. **100**, 9025–9031 (1994).

[48] X. Wu and B. R. Brooks, "A double exponential potential for van der Waals interaction," AIP Adv. **9**, 065304 (2019).

[49] A.-P. Hynninen (2017). "ap-hynninen/gse," Github. https://github.com/ap-hynninen/GSE

[50] X. Wu and B. R. Brooks, "Isotropic periodic sum: A method for the calculation of long-range interactions," J. Chem. Phys. **122**, 044107 (2005).

[51] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen, "A smooth particle mesh Ewald method," J. Chem. Phys. **103**, 8577–8593 (1995).

[52] S. Le Grand, A. W. Götz, and R. C. Walker, "SPFP: Speed without compromise—A mixed precision model for GPU accelerated molecular dynamics simulations," Comput. Phys. Commun. **184**, 374–380 (2013).

[53] S. Jo, T. Kim, V. G. Iyer, and W. Im, "CHARMM-GUI: A web-based graphical user interface for CHARMM," J. Comput. Chem. **29**, 1859–1865 (2008).

[54] S. Prasad (2025). "Apocharmm: A GPU-accelerated molecular dynamics package," Github. https://github.com/samarjeet/apocharmm (accessed 12 February 2025).