HPC Molecular Simulation Tries Out a New GPU: Experiences on Early AMD Test Systems for the Frontier Supercomputer

Ada Sedova*, Russell B. Davidson*, Mathieu Taillefumier†, Wael Elwasif*
*Oak Ridge National Laboratory, Oak Ridge, TN, USA
†ETH, Swiss National Supercomputing Centre (CSCS), ETH Zürich, Zürich, Switzerland
Corresponding email: sedovaaa@ornl.gov

***Abstract*—Molecular simulation is an important tool for numerous efforts in physics, chemistry, and the biological sciences. Simulating molecular dynamics requires extremely rapid calculations to enable sufficient sampling of simulated temporal molecular processes. The Hewlett Packard Enterprise (HPE) Cray EX Frontier supercomputer installed at the Oak Ridge Leadership Computing Facility (OLCF) will provide an exascale resource for open science, and will feature graphics processing units (GPUs) from Advanced Micro Devices (AMD). The future LUMI supercomputer in Finland will be based on an HPE Cray EX platform as well. Here we test the ports of several widely used molecular dynamics packages that have each made substantial use of acceleration with NVIDIA GPUs, on Spock, the early Cray pre-Frontier testbed system at the OLCF which employs AMD GPUs. These programs are used extensively in industry for pharmaceutical and materials research, as well as academia, and are also frequently deployed on high-performance computing (HPC) systems, including national leadership HPC resources. We find that in general, performance is competitive and installation is straightforward, even at these early stages in a new GPU ecosystem. Our experiences point to an expanding arena for GPU vendors in HPC for molecular simulation.**

***Index Terms*—high-performance computing, GPU programming, performance testing, molecular dynamics simulation**

I. Introduction

Molecular dynamics (MD) is a simulation method that propagates atomistic particles in time to expose important molecular fluctuations, traverse and sample potential energy surfaces, and predict physical observables useful for understanding molecular phenomena in physics, chemistry, and the biological sciences. Because MD simulations are limited to a maximum simulation-step size of about 0.5-1 femtoseconds (fs) for *ab initio* (quantum mechanical) methods or 2-5 fs for all-atom empirical force fields to avoid unacceptable drifts in simulation energy, and the targeted timescales to compare with many experiments may be on the millisecond timescale or greater, the top-performing MD programs used on high-performance computing (HPC) systems have made use of multiple types of parallelism to reduce the time per step, including threading models such as OpenMP, domain-decomposed parallel strategies with MPI, and acceleration with graphics processing unit (GPU) devices [1]–[8]. Some of these programs calculate an MD step in the range of 1 ms wall-clock time on systems of tens to hundreds of thousands of atoms [5], [6], [8], which is close to the performance roof-line set by the clock speed limit [9]. For simulations with empirical force fields, the force calculation that must occur with each step is the computational bottleneck [3], and for Born-Oppenheimer *ab initio* MD (AIMD) using density functional theory, it is the self-consistent-field energy calculation, from which forces are derived [7]. Beyond academic pursuits, both methods are extensively used in industry, from pharmaceutical research to materials science and engineering. For this reason, GPU vendors have an interest in supporting these types of codes for use on cloud servers and appliances such as NVIDIA's DGX.[1] In fact, recent container catalogs provided by NVIDIA for the DGX appliances have included a number of these molecular simulation programs,[2] and AMD has begun to develop its own such repository.[3] Here we report on the testing of four molecular dynamics programs: GROMACS [6], which uses C++ with MPI and OpenMP along with native device kernel code, Amber [10], which uses FORTRAN as its main programming language, together with native device kernel code, OpenMM [8], which uses C++ and native device kernel code, and CP2K [7], which uses FORTRAN, MPI, OpenMP, and C++ along with native device kernel code. All of these programs have previously used NVIDIA CUDA kernels and API as their device programming model for NVIDIA GPUs. The AMD versions of GROMACS, Amber, and OpenMM for AMD GPUs have been under development by AMD staff in collaboration with the code teams, and CP2K uses several different GPU kernels, which are in the process of being ported to HIP by researchers at the Swiss National Supercomputing Center (CSCS) together with AMD. The first three are codes

Notice: This manuscript has been authored in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

[1]https://www.nvidia.com/en-us/data-center/dgx-systems/
[2]https://ngc.nvidia.com/catalog
[3]https://www.amd.com/en/technologies/infinity-hub

that use empirical force fields (EFFs). They are used for biophysical simulations and condensed matter simulations of other large systems such as aqueous solutions and polymers. CP2K provides many different types of *ab-initio* simulation methods and different empirical methods. Here we focus on the linear-scaling self-consistent field calculation (LS-SCF) in CP2K [7], [11], [12], which is now able to perform AIMD on larger systems than are traditionally treated with quantum mechanical methods, including solvated (bio)polymers, which is an emerging capability.

II. HPC Systems and Programming Environments

We used the Summit supercomputer and the Spock system; both are housed at the Oak Ridge Leadership Computing Facility (OLCF). Summit is an IBM system containing approximately 4,600 IBM Power System AC922 compute nodes. Each node contains two IBM POWER9 processors and 6 NVIDIA Tesla V100 accelerators. Each processor is connected via two NVLINK connections each capable of a 25 GB/s transfer rate in each direction for a total of 100 GB/s bidirectional transfer.[4] Spock is an early-access testbed for the upcoming OLCF Frontier exascale supercomputer, and contains 36 compute nodes each with a 64-core AMD EPYC 7662 CPU and four AMD MI100 Instinct GPUs. The CPU is connected to all GPUs via PCIe Gen4, with 32 GB/s transfer rate in each direction for a total of 64 GB/s bidirectional transfer.[5] For both systems, and all four programs, the GNU Compiler Collection (GCC) was used. For compiling CUDA on NVIDIA, CUDA toolkit versions 10.2.89 and 11.2.1 were used. For compiling HIP on AMD GPUs, ROCm 5.0.2 was used. Spock is a Cray system which provides the Cray scientific libraries; these were used on Spock for linear algebra and FFTW for portions of the codes that use CPU-based routines. On Summit the FFTW module (which was found to perform on par with the the IBM Engineering and Scientific Subroutine Library (ESSL) version) and a combination of ESSL and Netlib Lapack and Scalapack were used (ESSL does not provide a complete implementation of Lapack). GPU-accelerated versions of scientific libraries are also used in these codes; ROCm's hipFFT and hipBLAS and NVIDIA's cuFFT and cuBLAS were linked for AMD and NVIDIA testing, respectively.

GPU specifications

TABLE I
Summary of official specification for NVIDIA V100 GPU and AMD MI100 GPU found on Summit and Spock, respectively[6,7]

	FP64 perf.	FP32 perf.	BW	GPU mem.
V100	7.8	15.7	100	16 GB HBM2
MI100	11.5	23.1	64	32 GB HBM2

FP64/32 perf: peak performance, TFLOPS, in double and single precision
BW: bidirectional CPU-GPU bandwidth, GB/s
GPU mem.: GPU memory

Table I shows a summary of the specifications published by each GPU vendor for their respective GPUs, including single (FP32) and double (FP64) peak performance, memory bandwidth and GPU memory (type and amount), for NVIDIA[6] and AMD.[7]

[4]https://docs.olcf.ornl.gov/systems/summit_user_guide.html
[5]https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html

III. Descriptions of Programs

Here we describe the four programs we tested in more detail, including the GPU acceleration schemes.

A. CP2K

CP2K is a large program that offers numerous different molecular simulation methods including multiple types of quantum mechanical methods [7], [13]. Of interest to researchers in condensed matter simulations for large, non-crystalline systems such as solutions, interfaces, and polymers is the LS-SCF method which allows scaling of the self-consistent field (SCF) across parallel compute resources and can enable MD simulations of thousands of atoms with quantum mechanical accuracy for hundreds of picoseconds. The distributed block compressed sparse row (DBCSR) library is a domain specific library used in CP2K that helps enable the LS-SCF routines and involves the use of an optimized small matrix multiplication (smm) library [12]. Several options exist for such a library within DBCSR, including the libxsmm CPU-based library[8] which targets Intel and other x86 CPU architectures with highly optimized vector-level optimizations. A GPU version of a smm library was developed within CP2K and is described below.

GPU acceleration: GPU acceleration can be used within the LS-SCF in several ways.

`libcusmm`: DBCSR uses the `libcusmm` library that deploys optimized smm kernels on the GPU. This library now uses just-in-time (JIT) compilation to optimize the smm kernels. On Summit, `libcusmm` is the primary way to run a performance-optimized smm for DBCSR and the LS-SCF method, as there is no version of libxsmm for the Power architecture. For AMD, this library is still under development and was not in a test-ready stage at the time of this writing: the JIT functionality is being re-worked to align with the rapidly changing ROCm software stack.

`PW-GPU`: The `PW_GPU` method is another GPU routine used in CP2K.[9] Although CP2K is based on multi-level real space grids, Fourier transforms are still used for various calculations. The fast Fourier transform is used for instance to merge all real-space grids contributing to the electronic density $n(\mathbf{r})$ together. CP2K applies the Fourier transform to each distributed grid and then adds Fourier components to the Fourier components of the largest grid before transforming it back to real space. The GPU implementation of this method (`PW-GPU`) can make use of one of two external libraries, cuFFT which is part of NVIDIA's cuda SDK, or hipFFT which is developed by AMD.

[6]https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf
[7]https://www.amd.com/system/files/documents/instinct-mi100-brochure.pdf
[8]https://github.com/libxsmm/libxsmm
[9]https://www.cp2k.org/howto:compile_with_cuda

Collocation and integration functionality (`GRID-GPU`): CP2K uses sparse matrices to represent physical quantities such as the potential or the density in a given Gaussian basis set; the density must be computed in real space to evaluate the potential along with other values. The inverse operation is called integration. If a plane wave basis is used instead, then these two routines are simple Fourier transforms.

The CP2K code relies heavily on the collocation and integration routines. These two functions are mathematically described by

$$n(\mathbf{r}) = \sum_{i,j} c_{ij} P_i(\mathbf{r}-\mathbf{r}_i) P_j(\mathbf{r}-\mathbf{r}_j) e^{\eta_i(-\mathbf{r}-\mathbf{r}_i)^2} e^{\eta_j(-\mathbf{r}-\mathbf{r}_i)^2} \quad (1)$$

for the collocation function and

$$c_{ij} = \int V(\mathbf{r}) P_i(\mathbf{r}-\mathbf{r}_i) P_j(\mathbf{r}-\mathbf{r}_j) e^{\eta_i(-\mathbf{r}-\mathbf{r}_i)^2} e^{\eta_j(-\mathbf{r}-\mathbf{r}_i)^2} d^3\mathbf{r} \quad (2)$$

for the integration function. i and j are composite labels that depend on the atom, angular momentum l_i and other parameters. The Cartesian harmonics polynomials P_i are expressed as a product of $(x-x_i)^\alpha$, $(y-y_i)^\beta$ and $(z-z_i)^\gamma$ with $l = \alpha + \beta + \gamma$ the angular momentum associated to a sub-block of the Gaussian basis set describing the potential of each atom i and j. In practice, the integration and collocation are limited to a finite interval that depends explicitly on the Gaussian parameters.

The collocation and integration routines can be cast in a series of matrix-matrix multiplications after discretization of the potential, density and Gaussian functions. While such approach works well on CPU, the current GPU implementation does not use this property because the coefficients matrix representing the density or potential in the Gaussian basis set are sparse and the resulting matrices of the Cartesian polynomials after discretization are small and irregularly shaped. In a new GPU version of these routines nicknamed `GRID-GPU`, optimized solutions to collocation and integration in CP2K have been developed for both NVIDIA and AMD by workers at CSCS.

In addition to these routines and the GPU-accelerated Fourier transform libraries, CP2K makes use of the linear algebra routines found within scientific libraries such as MKL, ESSL, or Cray Scientific Library. A number of domain specific libraries can also be used for various needs; here we limit the use of domain specific libraries to DBCSR. Here we tested CP2K version 10.0.

B. Amber

Amber is another large package that provides numerous different methods for molecular simulation [10]. While it includes a native implementation of a semi-empirical [14] and quantum mechanical methods [15], it is most widely used for molecular simulation with EFFs [16], [17]. These are physical potentials that are represented by simple analytical functions with scalar parameters that are fit to quantum mechanical calculations and experimental data. MD with EFFs is orders of magnitude faster than any form of quantum mechanical method but suffers from problems with generalizability; each new atom type or molecule must be parameterized before it can be simulated, which can be a painstaking process. Like the other EFF-based MD programs below, Amber has been developed primarily for the simulation of biomolecules, although a general EFF for simulating many types of non-biomolecular systems provided by Amber has extended its use to multiple applications including polymer science and more general solution-chemistry efforts. Amber is a long-standing program that has been in use for decades. Its base programming language is FORTRAN, with some newer C/C++ routines and device code (CUDA/HIP). Amber's performance strategy has not historically focused on vector-level SIMD or threading-based optimizations.

Amber's `pmemd` method uses the particle mesh Ewald (PME) approach for simulating long-range charge effects in periodic systems [18]–[20]; periodic systems are the primary method that MD uses to create an effective infinite system that mimics solvated, condensed phases and avoids the use of hard barriers to prevent the escape of particles. This program has been ported to the GPU and extensively optimized to make the most efficient use of a single GPU for large simulations that previously required many cores of a cluster. Communication between CPU and GPU is minimized to obtain this performance, at the expense of the ability to scale through MPI-based decomposition to multiple GPUs and compute nodes. However, single-GPU performance is now so good that most biomolecular systems of interest to researchers can be efficiently simulated with a single GPU. The focus for large, parallel simulation efforts is then shifted to the sampling of the energetic space of the system, a long-standing problem in MD, through the use of enhanced sampling methods that can achieve effective exploration of the phase space through the use of many parallel replicas [21], [22]. The Amber20 and AmberTools21 packages were tested here.

C. OpenMM

OpenMM [8] is another EFF-based MD program that also focuses on single-node performance. It provides a Python-based wrapper for launching simulations which interfaces with its C++/device code back end, which can also be used as a library without the Python component. OpenMM can make use of threading when running on the CPU, but it achieves substantial performance gains primarily from GPU acceleration, providing the first implementation of GPU acceleration for biomolecular MD within the programs tested in this paper in 2010 [23]. Also focusing on minimizing calculations on the CPU and back-and-forth transfers between host and device, OpenMM has provided performance that rivals Amber's implementation with a fully open-source program. The Python interface is used to provide object-oriented programming access to portions of the program not traditionally easily accessible to users, such as the creation of user-defined forces. These are compiled onto the GPU with its JIT compilation approach, which is also used in standard routines. OpenMM is also frequently used within an enhanced-sampling approach with multiple simultaneous replicas launched in parallel.

Here we tested OpenMM 7, using Anaconda-based installation from the respective conda-forge channels (openmm-hip[10] for AMD and openmm[11] for NVIDIA). OpenMM for HIP is also found on github,[12] as is the official OpenMM program.[13]

D. GROMACS

The GROMACS program [6], [24] is an open source package for EFF-based MD simulation that has traditionally focused on extensive performance optimization on multiple levels, as well as on the creation of a portable code that can make optimal use of many different resources. At the lowest level, hand-tuned vector instructions are provided for many different CPU architectures using a layered approach that enables an efficient addition of instructions for a new architecture using the upper level, which propagates down through the code and avoids extensive rewriting. Instructions have been provided for various x86 technologies, Power architectures, and ARM Neon and SVE extensions. Node-level threading is provided with OpenMP, and across-node parallelization with MPI. GPU acceleration has been supported through CUDA and OpenCL for close to a decade. The strategy for GROMACS has included maintaining the ability to scale to hundreds of compute nodes while simultaneously optimizing single-node performance as much as possible. While Amber and OpenMM now can simulate a million atom system on a single GPU, the total number of timesteps possible per day of simulation still lags behind what is achievable when using multiple GPUs or nodes on the same system; a factor of up to 10× speedup in this case is very useful for achieving optimal sampling more rapidly. However, in recent years GROMACS has offloaded increasingly more calculations to the GPU, and has been involved in single-node optimizations as well, which have recently begun to make use of GPU-direct technologies.[14] GROMACS developers are currently implementing large code structure changes which reflect some of these trends, along with the consideration of maintaining portability in the face of multiple emerging GPU technologies. It should be noted that the official GROMACS strategy for AMD GPU support is through hipSYCL,[15] therefore the AMD translation to HIP represents an unofficial version. We tested GROMACS version 2021, with the `mdrun` program.

E. Testing for single-GPU and small GPU-count performance

The Spock system is a small test system with 36 compute nodes and queue policies that limit the use of too many of these nodes at a time. Primarily for testing programs newly ported to AMD GPUs, the interconnects and related software and middleware do not represent what will be deployed on Frontier. Therefore, scaling performance is not a primary goal of the research presented here; results should also not be used to gauge the final performance of these programs on Frontier but rather to give a glimpse into the ongoing progress in deploying a new GPU architecture on an HPC system. CP2K and GROMACS still obtain significant performance gains from the use of multiple nodes; CP2K requires this for reasonable results due to the computational demands of quantum mechanical calculations. Other than these two programs, all benchmarks were limited to single node tests. For GROMACS we tested single-GPU performance for all benchmarks, and additionally single node, multiple-GPU runs for both ADH systems and with 2 nodes for the STMV input described below. The test system for CP2K's LS-SCF is one of the smaller examples of inputs for this routine, and the use of about a dozen nodes for this test thus represents a smaller test of performance equivalent to a single-node test for the three EFF-based programs.

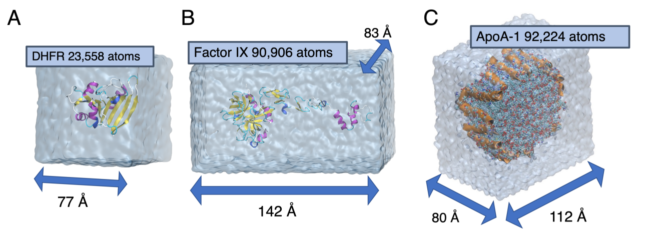

Fig. 1. Illustration of small and medium inputs for MD with EFFs, showing atom count and approximate system sizes. A: small system (∼25K atoms), B: medium system (∼90K atoms), C: another medium system (∼ 92K atoms) with a different protein and box shape. Explicit solvent molecules are represented as a water surface around the solute.

F. Inputs

Official inputs were used from each code's distribution, when available.[16,17] For testing GROMACS, we used the benchmark suite reported in [6] and available at Zenodo.[18] For the EFF programs, we tested a range of system sizes, including small and medium inputs typically simulated on single GPUs, along with two larger inputs that have only recently become possible to compute with single-node resources. Figure 1 shows some representations of the system sizes used for the small (about 25,000 atom) and medium (about 90,000 atom) systems. The dehydrofolate reductase (DHFR) system at 23,558 atoms represents the size of the small inputs and is the official small benchmark for OpenMM and Amber; GROMACS uses the ribonuclease (RNase) ZF-1a system which has 24,040 atoms. The medium-sized input is about 90,000 atoms. Amber uses the Factor IX protein with 90,906 atoms and OpenMM uses the apolipoprotein A-I (apoA-1) with 92,224 atoms. For a medium input, GROMACS provides the NADP-dependent alcohol dehydrogenase protein

[10]git revision ce22dbef84ec68aa910bbffed0f5e801e76ed9be
[11]git revision ad113a0cb37991a2de67a08026cf3b91616bafbe
[12]https://github.com/StreamHPC/openmm-hip
[13]https://github.com/openmm/openmm
[14]https://manual.gromacs.org/2022/release-notes/2020/major/performance.html
[15]https://enccs.se/news/2021/09/gromacs-adopts-hipsycl-for-amd-gpu-support
[16]https://ambermd.org/Amber20_Benchmark_Suite.tar.gz
[17]https://github.com/openmm/openmm/tree/master/examples
[18]https://zenodo.org/record/3893789#.YfR-7FjMJmc

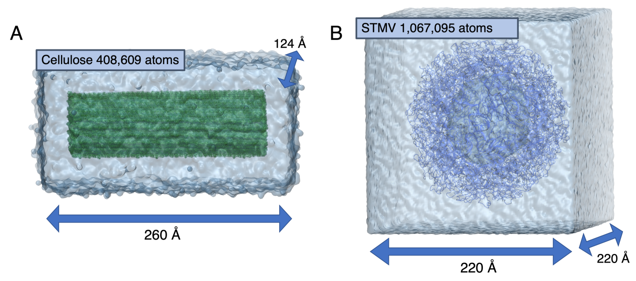

Fig. 2. Illustration of large inputs for MD with EFFs, showing atom count and approximate system sizes. A: cellulose system, ∼400K atoms and B: STMV system, ∼1M atoms. Explicit solvent molecules are represented as a water surface around the solute.

(ADH) in a trunctated dodecahedral solvent box with 95,561 atoms.

Large inputs, illustrated in Figure 2, include a 408,609 atom cellulose system, and a 1,067,095 atom satellite tobacco mosaic virus (STMV) system. A version of STMV exists for all three EFF programs. There is no input the size of the cellulose system found in the GROMACS benchmark set we used. In order to test the effects of using multiple GPUs for GROMACS on more inputs, we added a 134,177 atom version of ADH that is solvated in a cubic box and used both ADH systems and the STMV input with the MPI version.

For the EFF programs, we used official benchmark run parameters with the exception of the Amber inputs—here we removed one optimization, the `netfrc=0` flag, which turns off the net force correction that is used to correct energy drifts due to the smooth PME implementation, and is only advisable to use if strong thermal control is not required (without this flag the default value of 1 is used which turns on the correction).[19] We also added an NVT ensemble input to the Amber testing to produce additional data for comparison to OpenMM, for the large benchmarks. The OpenMM benchmark script allows for ensembles to be specified as input arguments, and we tested the NPT, NVT and NVE ensembles provided. All of the GROMACS inputs used the NVT ensemble. Within OpenMM's benchmarks, the program is initialized to activate the JIT compilation so that the output performance data does not include compilation time.

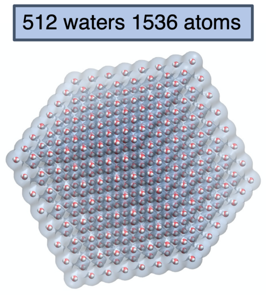

Fig. 3. Illustration of the water molecules system (1.5K atoms) used as input for CP2K.

[19]https://ambermd.org/doc12/Amber20.pdf

For CP2K, we used one of the LS-SCF regression tests for the LS-SCF[20] that includes a parameter that can be set to increase the size of the system through the replication of unit cells. This input generates a box of water molecules located at a set of points on a lattice. Figure 3 illustrates our input which used a replication number of 8, and contained 512 water molecules (1536 atoms). We added molecular dynamics to this input file using a 0.5 fs timestep, an NVE ensemble, and an initial temperature of 300 K.

IV. Ease of Building and Installation

We evaluated the building procedure and any problems that made building difficult. GROMACS and Amber use CMake/Make as a build strategy. The AMD versions of both programs now install successfully from source with minimum effort on Summit and Spock. OpenMM was installed successfully with Anaconda on both systems. CP2K currently uses Make and an in-house toolchain to assist with generation of the input `arch` file. Overall, this procedure was more difficult than installation of the other programs, but not more so than on any other HPC system such as Summit, due to the need to process many more user-defined build parameters and the potential to include a variety of third-party libraries.

V. Performance

Performance was measured using the provided performance output for each program. The metric used in the EFF programs is the simulated nanoseconds per day (ns/day), which must be combined with the MD timestep to be comparable across programs. For Amber and OpenMM benchmarks, the timestep is 4 fs, and for GROMACS it is 2 fs, therefore to compare GROMACS to Amber or OpenMM the value should be multiplied by 2. Reported values are the average over 3 independent runs. Variability was within 3 ns/day for all systems for all programs, except for the small system for all programs, which varied up to 5 ns/day. On Spock, performance was sensitive to NUMA placement and affinity; the `srun` parallel launcher within the SLURM scheduler provided identical control compared to manual control with `numactl` and additional AMD affinity flags.

For CP2K, the time per MD step depends on the convergence of the SCF calculation for that step. As the energy and position of the atoms varies, this time can also vary. CP2K uses values calculated during the first MD steps as an initial starting point for subsequent MD steps, and this speeds up SCF calculations after the first 5-19 steps significantly, after which the time per step usually stabilizes to within about 1 second. Here we report the average time per MD step over the last 10 seconds of our CP2K runs; all values had converged by this point.

In order to understand the scientific significance of the ns/day metric, it is important to consider the type of data that is being extracted from the simulation, which can vary. For comparison to many experimental values such as enzyme

[20]https://github.com/cp2k/cp2k/blob/master/tests/QS/regtest-dm-ls-scf-3/H2O-curvy-prop.inp

kinetics, large conformational changes, or binding events, long timescales (up to seconds or minutes) may be needed, which are not possible for direct MD simulations. However, strategic use of various theoretical approaches can help to overcome this barrier. For instance, for estimating protein-ligand binding affinities, methods such as free energy perturbation or thermodynamic integration can be used, which gradually modify forces on atoms across many replicas and can arrive at values that match experimental data [25]. Other replica-based methods, such as umbrella sampling, Markov state modeling, temperature replica exchange, and metadynamics, can use many parallel replicas to accelerate the sampling of the free energy space, and this information can be used within physical theories to provide long timescale data such as activation energies and rates [22], [26], [27]. Scaling of the sampling of the energy surface for weighted ensemble replica methods may be superlinear [28], meaning that a more effective sampling of the energy surface can be achieved using N replicas for M/N timesteps than by running a single replica for M timesteps. Other types of molecular events, such as thermal expansion and equilibration, vibrations, and smaller molecular conformational changes including solvent structure, can be simulated within pico- or nanosecond timescales. Microsecond to millisecond timescales are useful for determining residue-level dynamics and can also be used to compare to millisecond-scale values such as NMR relaxation data through correlation function based approaches. Therefore, for many tasks related to simulation of biomolecules and other polymers, obtaining over a microsecond of simulation time or more represents a breakthrough in the number of experimental values that can be predicted, and sampling equivalent to millisecond or second timescales via parallel replica methods is now enabling unprecedented insights, as well as the ability to fine-tune the physical models, which can now be better assessed knowing that any errors are not due to inadequate sampling of the space [29]. For the types of problems treatable with AIMD, the ability to simulate 5-10 ps/day for a system of several thousand atoms can lead to unprecedented ability to sample more rare events such as reactions and proton transfers, cooperative interfacial phenomena, and transport properties, and also opens the field to the study of conformation and solvent effects on reactions in larger systems such as polymers. Replica methods can also be applied to AIMD simulations, paving the way for nanosecond to microsecond scale simulation when deployed on HPC resources such as the upcoming exascale systems.

A. *CP2K LS-SCF Performance*

Figure 4 shows a comparison of performance for the CP2K benchmark on Spock and Summit, and the effect of the two GPU routines on performance on Spock. Because of the lack of the libxsmm library for Power CPU architectures and the in-progress status of GPU-DBCSR for AMD, exact comparisons are not possible. Nevertheless, performance on Spock is clearly better than on Summit. This is in part due to the highly optimized libxsmm. The new `GRID-GPU` method provides a 40% relative speedup over the CPU-only version, and the `PW-GPU` method provides an additional 20%. On Spock, scaling continues to 16 nodes, but on Summit, substantial scaling seems to stop at about 8-10 nodes. On Summit `PW-GPU` provides a comparable speedup to that on Spock (not shown), but this is still not enough to achieve the performance possible on the x86 system with GPU. The time per step demonstrated on Spock can provide about 7 ps/day if using a 0.5 fs timstep, and over 14 ps/day if using a 1 fs timestep.

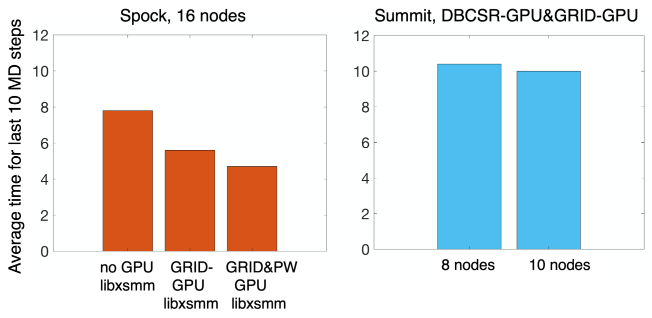

Fig. 4. Performance of CP2K water benchmark on Spock and Summit. On Summit, GPU-DBCSR was used along with GRID-GPU, with 6 ranks per node and 7 threads per rank. On Spock, DBCSR used the libxsmm library, GRID-GPU and PW-GPU, with 8 ranks per node and 8 threads per rank. Shown is averaged time over the last 10 MD steps in a run with 20 steps. Lower is better.

B. *Large inputs for Amber and OpenMM*

Figures 5 and 6 show performance results for the large inputs for Amber and OpenMM. The trends are similar, with the AMD MI100 outperforming the V100 for all ensembles and both large inputs. Amber performance is better than OpenMM for these systems, with the MI100 values for STMV without barostat at around 43 ns/day, and for cellulose about 110 ns/day, compared to about 30 and 83 ns/day for OpenMM.

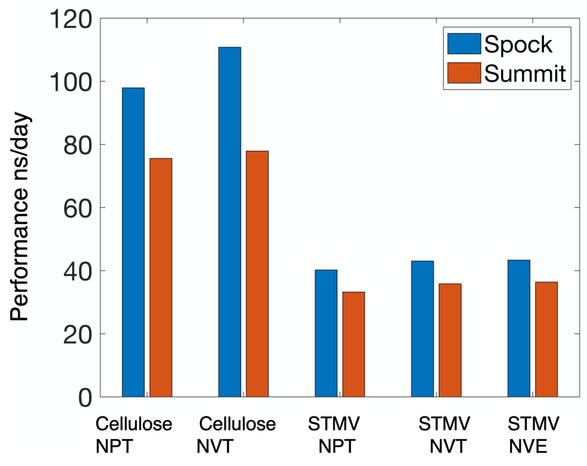

Fig. 5. Performance of large inputs for Amber on Spock and Summit, using one MI100 GPU or one V100 GPU, respectively. Simulations used a 4 fs timestep. Higher is better.

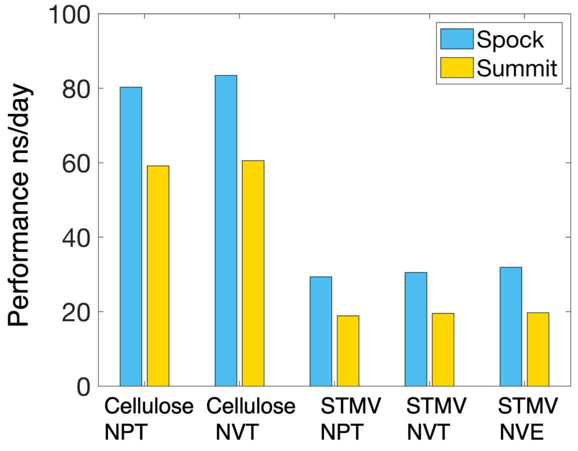

Fig. 6. Performance of large inputs for OpenMM on Spock and Summit, using one MI100 GPU or one V100 GPU, respectively. Simulations used a 4 fs timestep. Higher is better.

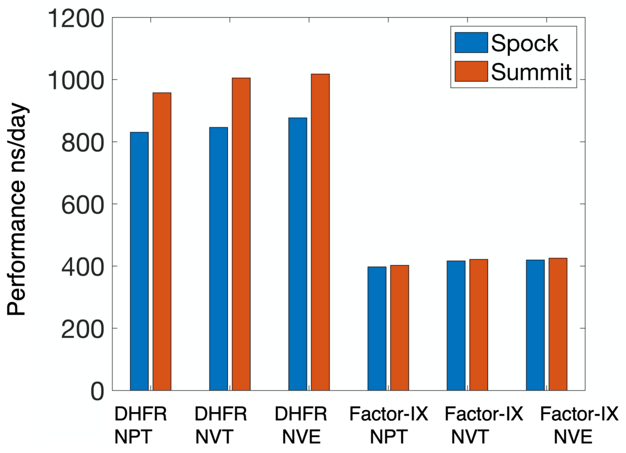

Fig. 7. Performance of small inputs for Amber on Spock and Summit, using one MI100 GPU or one V100 GPU, respectively. Simulations used a 4 fs timestep. Higher is better.

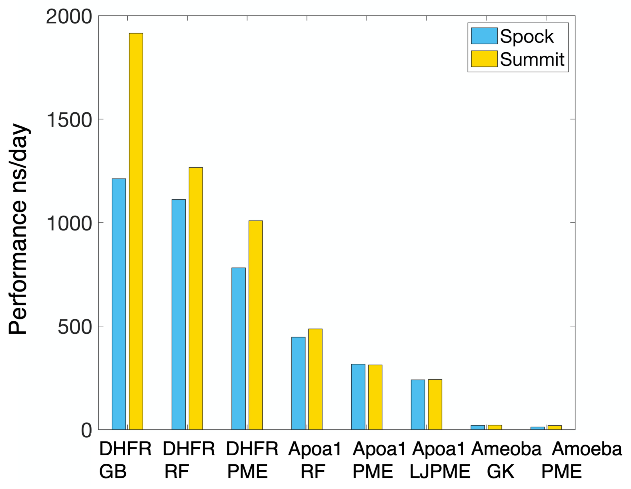

Fig. 8. Performance of small inputs for OpenMM on Spock and Summit, using one MI100 GPU or one V100 GPU, respectively. Simulations used a 4 fs timestep. Higher is better.

C. Small inputs for Amber and OpenMM

For the smaller inputs for OpenMM and Amber, the results are reversed, as seen in Figures 7 and 8. For OpenMM, the MI100 performance on Spock for these tests is approximately 80-95% lower than on the V100 on Summit for most tests, but comparable for the ApoA-1 PME tests, and below 65% for the DHFR generalized Born and surface area (GBSA) test (a continuum solvent model) and the AMOEBA (a polarizable model) PME test. It should be noted that these two methods are potentially not the first to be optimized in a new porting effort as the standard PME methods with fixed point-charge models are more commonly used and more in demand; it is therefore possible that these two calculations have not been treated with a similar optimization effort than the PME methods. The reaction field (RF) methods are similarly less commonly used. Nevertheless, the 77% relative performance of the DHFR PME test may point to a more general pattern.

For Amber, relative performance only drops to a low of about 84% for the NVT version of DHFR, but remains below 87% for all three ensembles tested for the small system. The medium system (Factor IX) test results in about 97% relative performance for all three ensembles. It is possible that smaller kernels are under-performing on the MI100, while larger kernels are performing well. It is also possible the that lower-bandwidth interconnect on Spock may contribute to some performance deficits.

D. GROMACS with and without MPI

As mentioned above, GROMACS makes use of OpenMP threading on the CPU, in addition to GPU acceleration, and MPI-based domain decomposition and multiple-GPU utilization. Despite an increasing amount of calculation ported to the GPU, we found that on Spock and Summit, CPU threading contributed non-negligibly to performance. An optimal number of 16 threads on Spock and 14 threads on Summit was found in all cases for the non-MPI build, except for the STMV benchmark on Spock which gained an additional 2 ns/day when using the entire node for threading (64 cores). In the case of the AMD EPYC 7662 "Rome" CPU, the deployment of a replica simulation on each of the four GPUs with the optimal 16 CPU cores for threading for each one would be possible. On Summit, however, the use of 14 threads would prevent the effective deployment of 6 replicas, and therefore a reduction in performance from the optimal setting, per replica, would be expected from the use of 7 threads per replica. Using 14 threads on Summit provided a substantial, 12 ns/day performance boost over the use of 7 threads for ADH-cubic, 60 ns/day for ADH-dodec, and 145 ns/day for the small RNase; the STMV test only gained about 2 ns/day performance when increasing from 7 to 14 threads. Figure 9 shows the performance of the no-MPI build on Spock and Summit, with 7 threads on Summit which would be the required settings to support a 6-replica run that made use of

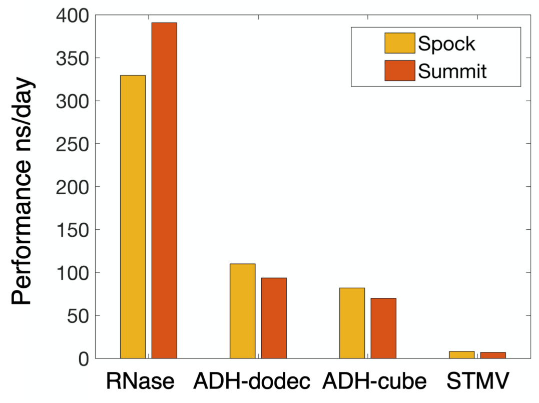

Fig. 9. Performance of GROMACS, single GPU (no MPI) version, on Spock and Summit, using 16 threads on Spock and 7 threads on Summit which is representative of a replica configuration that runs one simulation per GPU. ADH-dodec: dodecahedral version of ADH; ADH-cube: cubic version of ADH. Simulations used a 2 fs timestep. Higher is better.

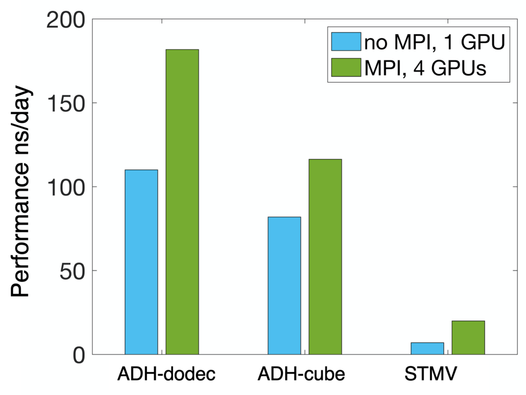

Fig. 10. Performance of GROMACS, MPI version, on Spock, using 4 ranks, 16 threads per rank for the ADH-dodecahedral (ADH-dodec) system and the STMV system, and 8 ranks, 4 threads per rank for the ADH-cubic system (ADH-cube). Simulations used a 2 fs timestep. Higher is better.

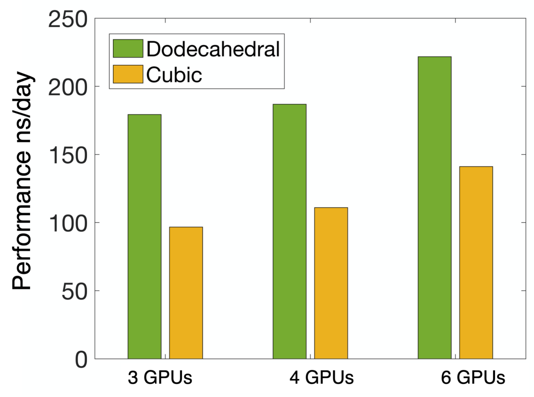

Fig. 11. Performance of GROMACS, MPI version, on Summit, using 7 threads per rank and an increasing number of GPUs. One MPI rank per GPU was used. Dodecahedral: ADH-dodecahedral system. Cubic: ADH-cubic system. Simulations used a 2 fs timestep. Higher is better.

all 6 GPUs on the node.

MPI for ADH and STMV inputs: Use of MPI and multiple GPUs provided a significant performance boost for the three tested GROMACS inputs (AHD-dodecahedral, ADH-cubic, and STMV). Figure 10 shows performance with the MPI version, using all 4 GPUs on the node, compared to the single-GPU version on Spock, illustrating the substantial performance gains that multi-GPU/multi-rank usage provides GROMACS. The performance for STMV, when the 2 fs timestep is accounted for, exceeds the performance of OpenMM for this benchmark. Using 2 nodes on Spock produced a result of 32 ns/day, which exceeds Amber's best result by about 20 ns/day when the smaller timestep is accounted for, demonstrating the usefulness of multi-node parallelization which can provide large boosts in amount of sampling possible per day for larger systems. However, the GROMACS STMV benchmark uses the reaction field method instead of PME for treatment of long-range forces, and thus direct comparisons may not be appropriate. On Summit, scaling past 3 GPUs with the STMV benchmark was difficult, potentially due to the reaction field method and other aspects of the input file; we have previously tested scaling of a 1.1 M atom system on Summit up to 40 nodes with a a more standard simulation scheme, achieving a final performance of over 100 ns/day. Figure 11 shows scaling across an increasing number of GPUs on a node of Summit for the ADH-dodecahedral and the ADH-cubic systems.

VI. Discussion and Conclusions

We have found that all 4 MD programs that have been ported to AMD GPUs were able to be built and run as expected on the Spock test system. Performance was comparable between the Spock test-bed and the production Summit system. For larger inputs, performance on Spock was better for Amber and OpenMM, while for small inputs, performance lagged. Reasons for this reduced performance could be due to higher kernel launch overheads and the effect of a lower bandwidth interconnect. For GROMACS, CPU threading was found to provide significant performance increases on both systems, with a much larger effect on Summit, for the non-MPI version; run configurations that can make use of a multiple replica deployment, important for enhanced sampling applications, would thus suffer more on Summit. The larger number of cores on Spock can support an optimal thread count for each task that runs on a single GPU. For this reason, performance of the two medium-sized inputs was higher on Spock, but for the small input, performance was again lower compared with Summit despite the use of 7 threads which was suboptimal. For CP2K, GPU acceleration of the GRID and PW routines provided a 1.7× speedup while the libxsmm library for x86 also helped boost performance to over 2× that of Summit.

Our findings present an optimistic view of the expected performance of these important programs on a new accelerator architecture that is breaking into the HPC domain, and promise exciting possibilities for scientific applications in molecular simulation on the Frontier supercomputer at the OLCF.

VII. Acknowledgment

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

We thank Vicky Tsang, Mazda Sabony, Leopold Grinberg, Anton Gorenko, Alfio Lazarro, David Case, Thomas E. Cheatham III, Szilárd Páll, Mark Abraham and Oscar Hernandez and their teams, for support, troubleshooting and valuable discussions.

References

[1] W Michael Brown, Peng Wang, Steven J Plimpton, and Arnold N Tharrington. Implementing molecular dynamics on hybrid high performance computers–short range forces. *Computer Physics Communications*, 182(4):898–911, 2011.

[2] W Michael Brown, Axel Kohlmeyer, Steven J Plimpton, and Arnold N Tharrington. Implementing molecular dynamics on hybrid high performance computers–particle–particle particle-mesh. *Computer Physics Communications*, 183(3):449–459, 2012.

[3] Ada Sedova, John D Eblen, Reuben Budiardja, Arnold Tharrington, and Jeremy C Smith. High-performance molecular dynamics simulation for biological and materials sciences: challenges of performance portability. In *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 1–13. IEEE, 2018.

[4] John Ossyra, Ada Sedova, Arnold Tharrington, Frank Noé, Cecilia Clementi, and Jeremy C Smith. Porting adaptive ensemble molecular dynamics workflows to the summit supercomputer. In *International Conference on High Performance Computing*, pages 397–417. Springer, 2019.

[5] Romelia Salomon-Ferrer, Andreas W Gotz, Duncan Poole, Scott Le Grand, and Ross C Walker. Routine microsecond molecular dynamics simulations with AMBER on GPUs. 2. Explicit solvent particle mesh Ewald. *Journal of Chemical Theory and Computation*, 9(9):3878–3888, 2013.

[6] Szilárd Páll, Artem Zhmurov, Paul Bauer, Mark Abraham, Magnus Lundborg, Alan Gray, Berk Hess, and Erik Lindahl. Heterogeneous parallelization and acceleration of molecular dynamics simulations in GROMACS. *The Journal of Chemical Physics*, 153(13):134110, 2020.

[7] Thomas D. Kühne, Marcella Iannuzzi, Mauro Del Ben, Vladimir V. Rybkin, Patrick Seewald, Frederick Stein, Teodoro Laino, Rustam Z. Khaliullin, Ole Schütt, Florian Schiffmann, Dorothea Golze, Jan Wilhelm, Sergey Chulkov, Mohammad Hossein Bani-Hashemian, Valry Weber, Urban Borštnik, Mathieu Taillefumier, Alice Shoshana Jakobovits, Alfio Lazzaro, Hans Pabst, Tiziano Müller, Robert Schade, Manuel Guidon, Samuel Andermatt, Nico Holmberg, Gregory K. Schenter, Anna Hehn, Augustin Bussy, Fabian Belleflamme, Gloria Tabacchi, Andreas Glöß, Michael Lass, Iain Bethune, Christopher J. Mundy, Christian Plessl, Matt Watkins, Joost VandeVondele, Matthias Krack, and Jürg Hutter. CP2K: An electronic structure and molecular dynamics software package - quickstep: Efficient and accurate electronic structure calculations. *The Journal of Chemical Physics*, 152(19):194103, 2020.

[8] Peter Eastman, Jason Swails, John D Chodera, Robert T McGibbon, Yutong Zhao, Kyle A Beauchamp, Lee-Ping Wang, Andrew C Simmonett, Matthew P Harrigan, Chaya D Stern, et al. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLoS Computational Biology*, 13(7):e1005659, 2017.

[9] Ada Sedova, Andreas F Tillack, and Arnold Tharrington. Using compiler directives for performance portability in scientific computing: kernels from molecular simulation. In *International Workshop on Accelerator Programming Using Directives*, pages 22–47. Springer, 2018.

[10] D.A. Case, K. Belfon, I.Y. Ben-Shalom, S.R. Brozell, D.S. Cerutti, T.E. Cheatham, V.W.D. Cruzeiro, T.A. Darden, R.E. Duke, G. Giambasu, M.K. Gilson, H. Gohlke, A.W. Goetz, R. Harris, S. Izadi, S.A. Izmailov, K. Kasavajhala, A. Kovalenko, R. Krasny, T. Kurtzman, T.S. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, V. Man, K.M. Merz, Y. Miao, O. Mikhailovskii, G. Monard, H. Nguyen, A. Onufriev, S. Pantano, R. Qi, D.R. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, J. Shen, C. Simmerling, J. Smith, J. Swails, R.C. Walker, J. Wang, L. Wilson, R.M. Wolf, X. Wu, Y. Xiong, Y. Xue, D.M. York, and P.A. Kollman. AMBER 2020, 2020.

[11] Joost VandeVondele, Urban Borstnik, and Jürg Hutter. Linear scaling self-consistent field calculations with millions of atoms in the condensed phase. *Journal of Chemical Theory and Computation*, 8(10):3565–3573, 2012.

[12] Ole Schütt, Peter Messmer, Jürg Hutter, and Joost VandeVondele. GPU-accelerated sparse matrix-matrix multiplication for linear scaling density functional theory. *Electronic Structure Calculations on Graphics Processing Units*, pages 173–190, 2016.

[13] Jürg Hutter, Marcella Iannuzzi, Florian Schiffmann, and Joost VandeVondele. CP2K: atomistic simulations of condensed matter systems. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 4(1):15–25, 2014.

[14] Ross C. Walker, Ichael F. Crowley, and David A. Case. The implementation of a fast and accurate QM/MM potential method in Amber. *Journal of Computational Chemistry*, 29(7):1019–1031, may 2008.

[15] Vinícius Wilian D. Cruzeiro, Madushanka Manathunga, Kenneth M. Merz, and Andreas W. Götz. Open-Source Multi-GPU-Accelerated QM/MM Simulations with AMBER and QUICK. *Journal of Chemical Information and Modeling*, 61(5):2109–2115, may 2021.

[16] Dario Vassetti, Marco Pagliai, and Piero Procacci. Assessment of GAFF2 and OPLS-AA General Force Fields in Combination with the Water Models TIP3P, SPCE, and OPC3 for the Solvation Free Energy of Druglike Organic Molecules. *Journal of Chemical Theory and Computation*, 15(3):1983–1995, mar 2019.

[17] Chuan Tian, Koushik Kasavajhala, Kellon A.A. Belfon, Lauren Raguette, He Huang, Angela N. Migues, John Bickel, Yuzhang Wang, Jorge Pincay, Qin Wu, and Carlos Simmerling. Ff19SB: Amino-Acid-Specific Protein Backbone Parameters Trained against Quantum Mechanics Energy Surfaces in Solution. *Journal of Chemical Theory and Computation*, 16(1):528–552, jan 2020.

[18] Tom Darden, Darrin York, and Lee Pedersen. Particle mesh Ewald: An N·log(N) method for Ewald sums in large systems. *The Journal of Chemical Physics*, 98(12):10089–10092, Jun 1993.

[19] Ulrich Essmann, Lalith Perera, Max L. Berkowitz, Tom Darden, Hsing Lee, and Lee G. Pedersen. A smooth particle mesh Ewald method. *The Journal of Chemical Physics*, 103(19):8577–8593, nov 1995.

[20] T. E. Cheatham, J. L. Miller, T. Fox, T. A. Darden, and P. A. Kollman. Molecular Dynamics Simulations on Solvated Biomolecular Systems: The Particle Mesh Ewald Method Leads to Stable Trajectories of DNA, RNA, and Proteins, 1995.

[21] Michele Invernizzi, Pablo M Piaggi, and Michele Parrinello. Unified approach to enhanced sampling. *Physical Review X*, 10(4):041034, 2020.

[22] Rafael C Bernardi, Marcelo CR Melo, and Klaus Schulten. Enhanced sampling techniques in molecular dynamics simulations of biological systems. *Biochimica et Biophysica Acta (BBA)-General Subjects*, 1850(5):872–877, 2015.

[23] Peter Eastman and Vijay S Pande. Efficient nonbonded interactions for molecular dynamics on a graphics processing unit. *Journal of computational chemistry*, 31(6):1268–1272, 2010.

[24] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C Smith, Berk Hess, and Erik Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1:19–25, 2015.

[25] Shunzhou Wan, Deepak Kumar, Valentin Ilyin, Ussama Al Homsi, Gulab Sher, Alexander Knuth, and Peter V Coveney. The effect of protein mutations on drug binding suggests ensuing personalised drug selection. *Scientific Reports*, 11(1):1–10, 2021.

[26] John D Chodera and Frank Noé. Markov state models of biomolecular conformational dynamics. *Current Opinion in Structural Biology*, 25:135–144, 2014.

[27] Gerhard Hummer and Jürgen Köfinger. Bayesian ensemble refinement by replica simulations and reweighting. *The Journal of Chemical Physics*, 143(24):12B634_1, 2015.

[28] Daniel M Zuckerman and Lillian T Chong. Weighted ensemble simulation: review of methodology, applications, and software. *Annual Review of Biophysics*, 46:43–57, 2017.

[29] Rafał Szabla, Marek Havrila, Holger Kruse, and Jiri Sponer. Comparative assessment of different RNA tetranucleotides from the DFT-D3 and force field perspective. *The Journal of Physical Chemistry B*, 120(41):10635–10648, 2016.