Porting Adaptive Ensemble Molecular Dynamics Workflows to the Summit Supercomputer ⋆

John Ossyra[1], Ada Sedova*[2[0000−0002−8233−3057]], Arnold Tharrington[2[0000−0002−2877−8768]] Frank Noé[3,4[0000−0003−4169−9324]] Cecilia Clementi[4,3[0000−0001−9221−2358]], and Jeremy C. Smith[5,1[0000−0002−2978−3227]]

[1] University of Tennessee, Knoxville, Tennessee, USA
[2] Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA
[3] Freie Universität Berlin, Berlin, Germany
[4] Rice University, Houston, Texas, USA
[5] University of Tennessee/Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA
*sedovaaa@ornl.gov

Abstract. Molecular dynamics (MD) simulations must take very small (femtosecond) integration steps in simulation-time to avoid numerical errors. Efficient use of parallel programming models and accelerators in state-of-the art MD programs now is pushing Moore's limit for time-per-MD step. As a result, directly simulating timescales beyond milliseconds will not be attainable directly, even at exascale. However, concepts from statistical physics can be used to combine many parallel simulations to provide information about longer timescales and to adequately sample the simulation space, while preserving details about the dynamics of the system. Implementing such an approach requires a workflow program that allows adaptable steering of task assignments based on extensive statistical analysis of intermediate results. Here we report the implementation of such an adaptable workflow program to drive simulations on the Summit IBM Power System AC922, a pre-exascale supercomputer at the Oak Ridge Leadership Computing Facility (OLCF). We compare to experiences on Titan, Summit's predecessor, report the performance of the workflow and its components, and describe the porting process. We find that using a workflow program managed by a Mongo database can provide the fault tolerance, scalable performance, task dispatch rate, and reconfigurability required for robust and portable implementation of ensemble simulations such as are used in enhanced-sampling molecular dynamics. This type of workflow generator can also be used to provide adaptive steering of ensemble simulations for other applications in addition to MD.

⋆ This manuscript has been authored 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.)

Keywords: High Performance Computing · Molecular Dynamics· Scientific Workflows · Adaptive Sampling.

1 Introduction and Background

Molecular dynamics (MD) simulations of condensed-matter systems require small steps in simulation-time, due to the accumulation of numerical error caused by a discretized model of a continuous physical process; classical simulations are limited to a maximum simulation-step size of about 2-5 fs for all-atom models, otherwise unacceptable drifts in simulation energy occur. The best-performing MD programs used for computational biophysics on High Performance Computing (HPC) systems calculate an MD step in the range of 1 ms clock time, which is close to the performance roof-line set by the clock speed limit. This constrains the timescale that a simulation can model, to a regime on the order of milliseconds. Obtaining simulation-times in the range of seconds or minutes, timescales relevant for comparison to experimental measurements, is therefore not possible.

To overcome this Moore's-law limitation on our ability to sample configurational space with physical accuracy, ensemble methods using many parallel simulations are increasingly being used [47, 57, 4, 25]. Using multiple parallel, short simulations throughout the simulation space, to reconstruct sampling equivalent to much longer simulations with statistical methods, is an approach frequently taken in MD simulations of condensed phases [22, 55, 27, 11, 53, 18]. Here we refer to these methods as enhanced-sampling molecular dynamics (ESMD) [9]. For example, algorithms derived from the fluctuation-dissipation theory of near-equilibrium statistical mechanics [26, 20] can be coupled with mathematical concepts from time-series and spectral analysis to determine rates and energetic barriers of physical processes [20, 55, 22]. The process of interacting with a ensemble simulation's progression to obtain optimal sampling of regions of interest is known as adaptive sampling [25, 21]. Markov-state methods use the assumption that the dynamics trajectory can be approximated as a Markovian process on some timescale [40]. The appropriate timescale can be determined by a statistical analysis of correlations of features of the system, chosen to best describe the system's evolution in time [10, 36]. Simulations can be launched from different points in the simulation space and new simulations can be started in regions that are under-sampled and important to the process of interest, based on additional statistical methods, in a progressively adaptive manner. If the final sampling performed by the simulations has been properly steered by this interactive analysis to cover the simulation space, a transition matrix can be assembled to report on transition rates between states. This Markov model can provide important long-timescale physical information such as kinetics, free energies, and dwell times of metastable states.

1.1 Scientific Workflow Platforms

Ensemble simulations require some type of workflow manager, or top-level program that manages parallel distributed tasks and then reduces the data into a

final form. For simple ensembles, home-made scripts can be used. However, as the workflow becomes more complex these scripts create impractical amounts of work and potentially can back-up shared launch-nodes and file systems.

For adaptive sampling MD implementations, the type of steering interaction used requires more complex solutions than simple checkpoint/restart management; new simulations must be initialized from correctly chosen conformations extracted from previous trajectories and the simulation program's starting files must be assembled in an automated manner. Additionally, records of the steering steps taken, analysis performed, and simulations launched throughout a complex ensemble workflow must be recorded as provenance metadata in a manageable format for later use, along with, potentially, more fine-grained logs of the compute components for benchmarking performance and debugging. Workflows should be able to support long-running heterogeneous applications such as multiple rounds of ensemble simulations combined with analysis tasks. State-of-the art molecular simulation requires that a workflow management program be able to integrate many separate jobs that each use, potentially, accelerators such as graphics processing units (GPUs), various levels of parallelism including threading and Message Passing Interface (MPI), along with multiple nodes each. Managing ensemble-workflows of these types of programs quickly becomes a non-trivial task. Along with ensemble-workflows to increase sampling of the conformational or energetic space, workflows to incorporate large datasets of either experimental data, or simulations at higher levels of theory, have recently become feasible.

Existing workflow programs have application programming interfaces (APIs) with different levels of complexity. General-use programs [2, 24, 16, 14, 13, 54] and some programs specific for ESMD [41, 7] have been developed. Radical Pilot (RP) [32, 52] is written in Python, and has been used on a variety of compute resources, including some HPC systems for which it must be explicitly ported by the developers. Recently, some general-purpose [31, 32] and some ESMD-based workflow programs [37, 7] have been able to scale to tens of thousands of cores on HPC systems. The Ensemble toolkit [8] runs on top of Radical Pilot, has recently added adaptable workflow functionality, and has been used on the Extreme Science and Engineering Discovery Environment's (XSEDE) resources and on the Oak Ridge Leadership Computing Facility's (OLCF) Titan computer, a Cray XK7 [6, 5]. However, none of these programs has been written specifically for pre-exascale and exascale use. As HPC systems approach exascale, we may find that some solutions may fail on these machines.

HPC-Specific Solutions Groups in many areas of computational science are realizing that a need for HPC-specific workflow managers exists. There is a need for well-supported workflow programs that can handle account allocations, schedulers, and multiple levels of parallelism within each component, and can scale to thousands of nodes (hundreds of thousands of cores).

To address this need, the Swift/T dataflow parallel scripting language and runtime system was created to allow for the construction of large, scalable workflows on HPC systems [56]. The original Swift implementation was able to achieve

a task-dispatch rate of 500 tasks launched per second, but these were dispatched from a single node, and thus the program was limited by the memory of that node. To overcome this limitation, inclusion of the Turbine engine allowed for the control of the program to be coordinated by several nodes, relieving these memory constraints and promoting scalability [56]. With this improvement the task dispatch rate exceeded 60 thousand tasks/sec and Swift/T was able to use 64 thousand cores at greater than 90 % efficiency.

However, the Swift/T platform will not persist over multiple instances resource allocations, for example, over several jobs that use the (often short) allowable job time limits on many HPC systems such as at the OLCF. Furthermore, there is a limitation of fault tolerance handling, and hence the entire workflow may terminate when one task fails [15]. Other HPC-specific workflow efforts include those at the Livermore National Laboratory, focusing on development of workflow managers that can provide experimental data analysis and visualization, and incorporate some simulation as well [28], and initiatives within the Exascale Computing Project [33] such as the Advanced Technology, Development, and Mitigation (ATDM) software technology effort, to develop a variety of solutions to support data management and workflows, primarily in nuclear science. However, these developments have not yet provided an HPC-based solution directly usable for adaptive enhanced-sampling workflows in MD.

Performance Portability Over HPC Systems Performance portability is essential for longevity of software products; with Department of Energy systems, architectures are cutting-edge and vary significantly between generations. Therefore, strict protocols for software design must be maintained to obtain robust, efficient and portable computational solutions [44, 45, 35, 51]. Portable software design for HPC should include standard, well supported languages, modular, linear design, and simple construction [44, 45]. Recent solutions for scientific workflow management systems have involved the use of a number of programming languages, and heavily nested programs, including extensive use of Python. Testing these solutions on the OLCF Summit IBM Power System AC922 computer will expose potential portability problems with programs that were deployed on previous HPC systems, thus facilitating a portable solution for adaptive MD workflows on future HPC systems.

Use of Database Management Programs The use of database management systems within scientific workflow platforms is not a new concept; in 1998 Ailamaki and co-workers noted that a database management system (DBMS) has many essential properties of a workflow management system and with a data-object view of scientific workflows the DBMS could be effectively used to manage related parallel computational tasks [3]. The performance and variety of DBMS implementations have grown significantly in recent decades, and non-relational (NoSQL) databases have been found to perform better than relational (SQL) ones in a number of areas [34, 29]. Recently several scientific workflow management solutions have incorporated both SQL and NoSQL DBMSs into their software design. For instance, Copernicus uses the Python interface with sqlite, and Fireworks [24] and RP use MongoDB for task management. The use

of DBMS in scientific workflow programs has been discussed in recent literature within the context of HPC and massively parallel distributed solutions [49].

Current Work Here we test the performance and portability of a workflow management program driven by a DBMS on HPC systems. This was implemented first on the Titan supercomputer with the adaptive sampling method within ESMD as a use-case. With this method we use inverse-count sampling [21] from a Markov model built with a TICA [43] and PCCA++ [23] analysis pipeline. The program, written in Python, has a database-centric architecture and uses the MongoDB NoSQL DBMS for status synchronization. The use of the DBMS allows for reliability and fault tolerance while the use of NoSQL provides a more flexible, distributed and scalable solution [46]. Here, we discuss the successive porting of components of this program to the Summit computer and testing for function and performance.

2 Building an Adaptive Ensemble-Simulation Program

We describe the implementation and performance testing of a Python program designed to create HPC-based scalable workflows for adaptive sampling of MD simulations. To run reliably and independently over possibly weeks to months, which can be typical of many use-cases, the program was designed as a distributed application that can run from a laptop or directly on an HPC resource and can automate asynchronous workflow creation and execution. The program, available at `https://github.com/markovmodel/adaptivemd`, is an extension of a previous version of the code that was implemented [38] on smaller systems, and used programmatic elements from the "OpenPathSampling" program [39, 50].

The program consists of a Python layer that interacts with a MongoDB database via the PyMongo interface [42]; worker units called executors use Python subprocesses to execute tasks. The database functions as a tool for a Python-based distributed task scheduler: as a peer-to-peer data sharing platform, and as a database for provenance information, task input data and reduced output data. The distributed task model could also allow for the use of different computer architectures for different computational tasks, where each may perform more optimally. Fields a user provides allow the program to bind resource-specific properties to the task descriptions at run-time. Several resources can be configured to be used simultaneously in this distributed computing model, and tasks can be targeted to any available resources.

While the program has an API that can be used to manage a number of different kinds of simulations and analysis or steering methods, our currently implemented use-case seeks to improve the sampling of the slowest processes in a biomolecular simulation, using unbiased MD replicates and Markov-state modeling. The intent is to allow adaptive sampling procedures to either be fully automated or to proceed with varying amounts of user interaction at a number of levels, including, potentially, run-time steering. Run-time adaptions can be made to 1) task properties such as analysis type or parameters, 2) workload

properties such as task count, or 3) workflow properties such as convergence criteria. Using its native executor class, the program can address the entire chain from a workflow-generating instance down to task execution.

```
def generate_workflow(length, loops):

    for _ in range(loops):
        restart_frames = sampling_function(project.models.last)

        trajs = project.new_trajectories(restart_frames)
        model = modeller.execute(*margs)
        tasks = [t.run() for t in [model]+trajs]

        project.queue(tasks)

        yield any([t.is_done for t in tasks])

    project.workers.command("shutdown")


project.add_event(
    generate_workflow(n_md_steps, n_workloads)
)
```

Listing 1.1. A simple workflow generating function that runs a fixed number of workloads with MD replicates of a fixed length. This generator can be written by a user (more sophisticated workflow generators are provided in the program) and given to the project event loop.

Listing 1.1 illustrates a simple workflow generating function that runs a fixed number of workloads with MD replicates of a fixed length. This generator can be written by a user (more sophisticated workflow generators are provided) and given to the project event loop, and must yield functions that return Boolean values. At runtime, the current Boolean function will be executed repeatedly in the event loop value until it evaluates to True, in this case when any task completes. On this event, the application will iterate the generator to yield a new function, executing a code block in which additional tasks are queued in the project and specified in the new progress function.

Our aim here is to test the scalability, robustness, and performance of this method on large HPC resources at the OLCF. To keep configuration and installation simple while providing reliable workflow execution on a wide range of computational resources, we have separated the workflow platform into hierarchical levels of complexity, and create removable interfacing with other Python-based workflow programs. The current implementation can function with or without an interface to RP.

Unlike a single large job that runs on an HPC system, workflows involve many initialization times over the course of the program. This initialization overhead may have a non-negligible impact on overall performance of the workflow and must be optimized. File accesses, such as accessing executables and task descriptions, must be handled efficiently when they occur repeatedly through the workflow. For example, launching many concurrent instances of an executable involves a decision about whether all executors access the same executable file, receive copies of the file, or access the file from the database. This decision may have to be made for many different executables and/or analysis tasks that may need to access the same copy of a Python library. The optimal decision will vary

based on the task, the number of executors, and the size of files transferred, which may include larger data files and a library of dependencies. The optimal solution for a particular part of the workflow may vary by HPC system as well.

2.1 Using a Database Program to Support Interactive, Adaptable DAG-based Task Scheduling, Analysis and Provenance

For adaptable and steered simulation workflows, we would like to allow as little or as much manipulation of the directed acyclic graph (DAG), which represents the workflow, as desired, including possibly an on-the-fly interaction which would result in a constantly-adapting DAG. The database queuing functionality provides the potential for run-time steering of complex workflows: the database program facilitates multiple live connections who query a consistent state. This transfer of information allows for on-the-fly receipt of reduce operations, sampling from the database and the tasks in the workflow, and synchronization of new tasks to be executed, by executors who are distributed on arbitrary resources.

An immediate, reliable, and fault-tolerant utilization of tasks is therefore facilitated by the database program, as is the ability to increase file accessibility and redundancy. The DBMS provides the requirements to perform task-to-data targeting; code can be added that queries a task property and accepts or rejects a resource assignment.

Provenance data is an important consideration for workflows with many tasks [12, 48]. The MongoDB dataflow collections provide a reliable mechanism for storing and managing our queues of tasks and their metadata. This metadata can facilitate the management of provenance information. In our usage, the database contains mainly the metadata about what happens, using a "pretask, main task, and posttask" abstraction.

2.2 Use of Radical Pilot

To test if the RP intermediate job manager can provide more robust workflow management, we created a version of the program on OLCF Titan which integrated with RP. This tool can enhance run-time error detection and correction functionality. However, it has a much higher installation and configuration overhead if not already available on a user's resource: it requires the developers to specifically configure the program for a particular HPC system. Results on Titan with and without the RP interface are reported below.

2.3 Experiences and Performance on OLCF Titan

A complete Markov-model ESMD workflow with TICA and PCCA++ analysis was deployed on the Titan Cray XK7 computer at OLCF, using the OpenMM simulation engine [17]. The program used the following versions of key programs: Mongo 3.3, PyMongo 3.6, Python 2.7.13, PyEmma 2.4, Numpy 1.15, OpenMM 7.0.1, and CUDA 7.5. Titan uses the Cray Linux Environment as its operating system. Titan nodes each contain a 16-core 2.2GHz AMD Opteron 6274 (Interlagos) processor and NVIDIA Kepler K20X GPU.

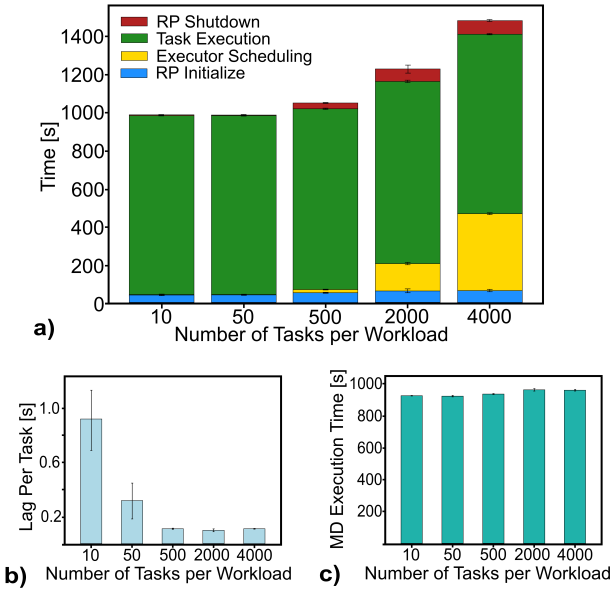

Fig. 1. Parallel performance of ensemble simulations on Titan, using Radical Pilot (RP) interface. Single round of tasks, 3 replicates, 1 task per node. Tasks were molecular dynamics simulations of small proteins using the OpenMM [17] MD program. a) Time for each component of the workflow. b) Total non-execution time (lag) divided by total tasks c) Time for task execution only, for varying workload sizes.

Fig. 1 illustrates performance of the workflow program with the RP interface, including the components of different parts of the workflow, and the latency (non-task time) divided by number of tasks. This normalized latency reports on the cost-per-task of the workflow program's overhead. The program was successfully implemented with and without the RP layer. Without RP, up to 5000 parallel, single node simulation tasks were executed with greater than 90% weak scaling efficiency (data not shown), but we were unable to execute use-case workflows including analysis tasks at this scale: our program deploys a number of executor units, and due to the large number of executor-associated connections to the database, and on Titan, network congestion and high hardware utilization made the database host unreliable in transferring the analysis output data with $O(10^3)$ tasks. To improve the network congestion of the stand-alone program, a splitting of traffic to the database server could be accomplished by the use of sharding. In addition, accessing the shared Python libraries with multiple tasks at start-up frequently resulted in a failure to find the file handles for the programs executed by tasks. This seemed to be caused by conflicting file reading attempts, and was mitigated by a spacing out task launch times by small time increments.

A reduction in sensitivity to analysis-task synchronization was found when using RP. A single RP Agent replaces the program's executor reads from the database, reducing connection-associated network congestion and overhead on the host. The DBMS interactions are replaced by writing to local files. RP builds an execution script from each task's instructions, and replaces the spaced task-

launch procedure with a uniform initialization time. We did not observe filesystem errors at high scale, however the scaling of the RP version of the program stops at about 8000 nodes on Titan, at which point the RP program crashes with an out of memory error for the array used to keep track of the tasks. This limitation is currently being addressed by the RP developers. **Figure 1.a** shows weak-scaling performance using Radical Pilot to execute up to 4000 single-node tasks. The total workflow duration is shown partitioned to 4 duration intervals. The time to schedule, then dispatch, tasks, as well as latency between termination of the final task and workload termination, are seen to increase with scale. **Figure 1.b** shows that these scheduling and shutdown latencies combine to a per-task cost that stabilizes to about 0.1 seconds per task with 500 or more tasks. Initial bootstrapping of the environment is seen to be independent of the number of tasks, and **Figure 1.c** shows that the main simulation task duration is also largely independent of scale.

3 Experiences and Performance on OLCF Summit

We approached the transition from Titan to Summit by testing isolated components before then testing several executables run by tasks within the program. OLCF Summit consist of 4608 IBM Power System AC922 nodes. Each node contains two IBM POWER9 processors and six NVIDIA Volta V100 accelerators. Summit uses Red Hat Enterprise Linux Server release 7.6 (Maipo) as an operating system. Currently RP is still being ported to Summit by the developers, so here we can only consider building and testing the program's stack without RP. We tested the performance of two MD programs to determine the optimal resource configurations for different sized biomolecular systems. Then after testing system functionality when scaling to thousands of simultaneous executables, we test the program itself. With the success of each component, we can interpret the contributions of each to the performance of the workflow program on Summit.

3.1 Python packages on Summit: Scientific Libraries

Although Python is not traditionally used for parallel programming, HPC centers are increasingly seeing a large amount of Python usage as a wrapper-level "glue" for compiled libraries, especially for machine learning [30]. While Python is often referred to as a portable, ubiquitous language, on Summit we have noted several challenges in porting portions of our Titan workflow. Many Python packages contain inner regions of code written in C/C++ for performance, and also may require calls to additional libraries. On Summit, we found that the NumPy (`https://www.numpy.org/`) configuration interface does not support the native Summit Engineering and Scientific Subroutine Library for linear algebra. A version of OpenBLAS (`http://www.openblas.net/`) is available in a Power8 version and also in a PowerPC version, but there is no version of BLAS (`http://www.netlib.org/blas/`) optimized for Power9 that can be incorporated into NumPy easily. In addition, for future incorporation of RP, a Python virtual environment must be used, and we have found that a pip installation

of numPy using a `virtualenv` is not able to find the Power8 version of OpenBLAS. The analysis portion of our workflow uses the Python package PyEMMA for Markov modeling [43]. This package in turn uses the MDTraj Python package (`http://mdtraj.org/1.9.0/`). MDTraj contains several libraries written in C/C++ that contain hard-coded x86 single instruction, multiple data (SIMD) vector intrinsic functions, and therefore, the MDTraj package and PyEMMA cannot be built on Summit without manual removal of these SIMD-containing portions. Without MDTraj, streaming "load iterators," of large trajectory files cannot be utilized, and other essential processing tools for the OpenMM program become unavailable. Due to our distributed computing capabilities, we are able to run the analysis tasks on another computer, but this can become inconvenient when the MDTraj frame-finding tool cannot be used within a task running on Summit.

3.2 Single-node Performance of Two State-of-the-Art Open-Source MD Programs

Measuring the single-node performance of the underlying MD programs used as the "engine" in the ensemble simulation is important for determining which program and configuration will be used in the workflow for a particular biomolecular system. These choices may vary with system size, and the requirements for individual simulation lengths which are ultimately dictated by the system kinetics and the resources available.

Table 1. Single-node performance for OpenMM on Summit. Sim-time: simulation time, perf: performance.

Size	**Sim-time (ns/day)**	**Time (ms/step)**	**Perf. (steps/sec)**
29 K	292	0.6	1690
134 K	97	1.8	561
1 M	11	15.7	64

While the workflow program we are developing can use any executable, here we investigate single-node performance of two open source MD programs that are of the best-performing for systems under 1 M atoms, OpenMM [17] and GROMACS [1]. We used OpenMM version 7.3, GROMACS 2018.3, CUDA 9.2, and gcc 6.4, with Spectrum MPI version 10.2. These programs both make heavy use of the GPUs, and GROMACS also uses several layers of parallelism on the CPU: MPI, OpenMP, and SIMD. GROMACS allows the user to specify if the Particle Mesh Ewald (PME) calculation should take place on the GPU or the CPU. Single-GPU GROMACS calculations used 1 MPI rank when PME was calculated on the CPU, and 2 ranks when PME was on the GPU. 6-GPU version used 6 MPI ranks. All simulations used 2 fs timesteps, and the NVT ensemble. OpenMM tasks used 1 GPU and 7 CPUs.

Tables 1, 2 and **3** display single-node performance of these two programs on Summit. We found that on Summit, these programs perform one MD step

Table 2. Single-node performance for GROMACS on Summit, One GPU. First value: PME calculated on CPU, Second value: PME calculated on GPU. Sim-time: simulation time, perf: performance.

Size	Sim-time (ns/day)	Time (ms/step)	Perf. (steps/sec)
29 K	284/253	0.6/0.7	1644/1464
134 K	64/56	2.7/3.1	370/324
1 M	8/6	21.6/28.8	46/35

in sub-millisecond compute times on a single node, for systems with tens of thousands of atoms. GROMACS obtains under 2 ms per step for systems of around 100 K atoms on a single node, and close to 20 ns/day for a 1 M atom system on a single node, using 2 fs simulation-time integration steps. This is a result which previously was only attainable using hundreds of nodes on Titan. As a result, with adaptive sampling effective timescales of tens of milliseconds within a few weeks of simulation will be possible.

Thus it is possible to, for instance, run close to 24 thousand simultaneous MD tasks with a system of about 130 thousand atoms using OpenMM, with 6 tasks per node, and obtain close to 100 ns/day, providing for excellent sampling within the MSM method. One could choose to use half of a Summit node, or an entire node, with GROMACS, and obtain about twice the simulation time, per day, *versus* OpenMM, for a system of around 30 K atoms or 1 M atoms. The choice of whether to run six times more tasks, or aim for longer simulation times per day, per task, will depend on the biomolecular system and the conformational space that must be sampled.

Table 3. Single-node performance for GROMACS on Summit, 6 GPUs. First value: PME calculated on CPU, Second value: PME calculated on GPU. Sim-time: simulation time, perf: performance.

Size	Sim-time (ns/day)	Time (ms/step)	Perf. (steps/sec)
29 K	231/423	0.8/0.4	1337/2448
134 K	66/110	2.6/1.6	382/637
1 M	11/18	15.7/9.6	64/104

Using Several Nodes Per Job With GROMACS Unlike OpenMM, GROMACS uses MPI. The implementation allows for linear scaling over tens of thousands of cores for larger systems. It is therefore possible to perform ESMD on systems containing several million atoms. On Summit, the simulation of 1 M atoms should be efficient on even a handful of nodes. We found the most efficient workflow configuration for running many parallel simulations of a 1 M atom system when using the GPU-offloaded PME calculation was three nodes per simulation instance. After this number, the GPU-based PME calculation suffers from high communication overhead, and performance is greatly improved by moving the PME calculation to the CPU.

Figure **Fig. 2** shows scaling of the 1 M atom system over 16 Summit nodes, using the GPU-based PME. Without GPU-PME, 100 ns/day can be achivied

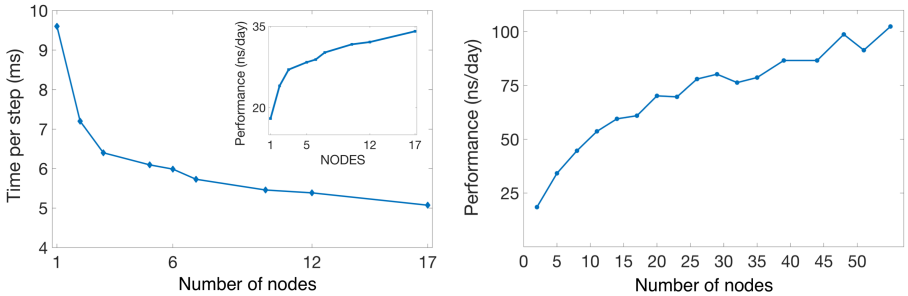

Fig. 2. Performance of GROMACS on Summit, 1 M atom system. Left panel: Using GPU offloading of the PME calculations as well as the non-bonded forces. Right panel: Without PME offloading. Configuration used 6 GPUs and 6 MPI ranks per node, with 7 OpenMP threads per MPI rank. PME offloading quickly becomes inefficient but can provide a good solution when using about 3 nodes per task. GROMACS scales linearly to much higher node counts when PME offloading is not used.

using 50 nodes. It should be noted that on Titan the same sized system achieved this performance using 1024 nodes. For ensemble MD, depending on available resources and timescales required, many tasks each using small numbers of nodes, or fewer tasks each using dozens of nodes could be performed.

3.3 Performance of Allocation Management Tools: Running Thousands of Small Concurrent Workflow Tasks

Here we describe tests of the allocation management system along with other aspects of system performance on Summit when running thousands of parallel tasks without an MPI programming model. This is relatively non-standard, historically, for HPC computing. However, the feasibility and appeal of ensemble simulations on these large systems is increasing, and thus this model is becoming more popular.

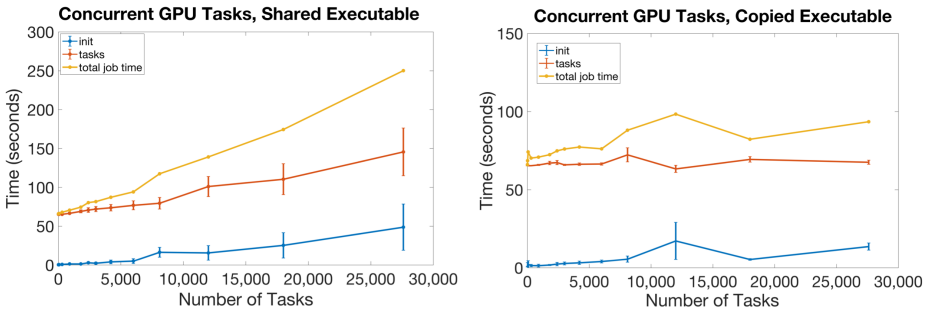

Fig. 3. Performance of concurrent tasks launched from a single `jsrun` on Summit, CUDA/C executable. Left panel: a single executable was read by all processes. Right panel: executable was copied and each process read its own file. Copy time not included in total job time. Tasks were run 6 per node.

There are two configurations for launching multiple jobs within our workflow program. A single `jsrun` command (the IBM Job Step Manager tool for launching jobs) can launch thousands of instances of an executable using multiple resource sets, or thousands of tasks can each call a different instance of `jsrun`. Neither situation is commonly anticipated in the optimization of large HPC computers. In the following we describe the performance and stability of the Summit software stack under both configurations, without the workflow program.

Single `jsrun` Running Thousands of Tasks Using a single instance of `jsrun`, up to 27600 simultaneous executions of simple programs were performed, using 6 instances per Summit node. This utilized up to 4600 of the 4608 nodes on Summit. **Figures 3** and **4** show results from these tests, using a CUDA/C executable or a C executable with OpenMP directives. For the latter, we used the HACCmk cosmology microkernel [19]. In contrast to on Titan, we found that on Summit no errors were encountered while attempting to access the same executable by multiple processes, as there were on Titan. However, this procedure resulted in a large initialization overhead. In addition, in this situation the execution times of the tasks increased for high node counts (**Figure 3**, left panel). **Figure 4** shows performance of CPU executables, without threading (left panel) and with OpenMP threading (right panel). Total job latency was similar between the two versions. In both cases, for over 20,000 concurrent executions, job latency was appreciable and similar, and less than for the GPU executable. This difference may be a result of system fluctuations, however, such as batch node traffic. In all cases, the latency was significant and could cause job time to be more than double that of task time. However, with long running tasks, these effects become less noticeable.

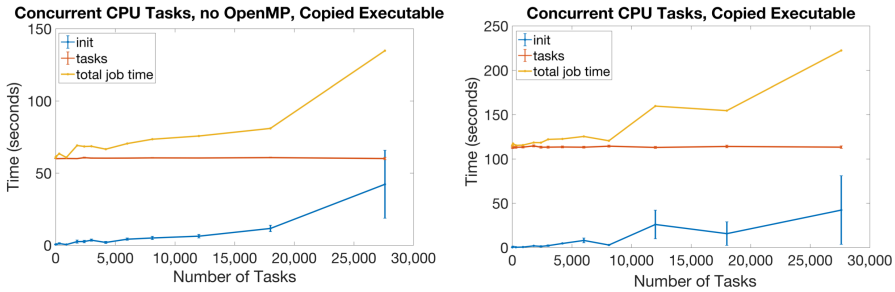

Fig. 4. Performance of concurrent tasks launched from a single `jsrun` command on Summit, CPU executable. Left panel: C executable, no OpenMP or other threading in executable. Right panel: OpenMP-containing executable run with 7 threads. Tasks were run 6 per node. Total task time for single executable is, left: 60 seconds, right: 113 seconds.

Thousands of Parallel `jsrun` Instances We found that on Summit, thousands of concurrent instances of the `jsrun` tool were not allowed due to process limits set on the batch nodes. This behavior prevents more than 1000 tasks from

being launched simultaneously from 1000 `jsrun` calls. Unfortunately this prevents the scaling of a workflow to all of Summit when using MPI-containing executables and single-node jobs, because they cannot be launched inside the same `jsrun` instance or they will couple. Therefore under the current circumstances the scaling to all of Summit of GROMACS inside of the workflow program would require the use of multiple nodes per simulation.

3.4 Testing the Adaptive Workflow with Executables Containing CUDA and OpenMP

Here the full workflow program was tested using executables containing CUDA, and C with OpenMP. MongoDB is a document database and thus each communication requires opening of a file. We used Mongo version 3.6.11 with the ppc64le build, PyMongo 3.7.2, Python 3.7.3, and NumPy 1.15.4.

Fig. 5. Timelines of workflows for 60, 300, 1800, and 4200 tasks. Due to the short duration of tasks used here, under 1000 executors stay busy as previous executors pick up tasks after they finish. At around 2000 tasks, incorrectly inherited `ulimit` settings limit the number of file handles allowed and tasks are rejected, however, all required tasks are completed.

We discovered a bug in the Summit IBM Spectrum Load Sharing Facility (LSF) batch scheduling system's configuration that propagates the wrong environment inside of a `jsrun` instance: while the hard maximum limit for number of file handles per process (`ulimit -n`) is set to around 65 K on the compute nodes, the `jsrun` environment incorrectly inherits the settings from nodes with a limit set to 4096. The database program requires two files opened to support a connection to each executor. Therefore, for concurrent launching of 6 tasks per node (run by 6 executors) using all of Summit, about 54 K files need to be opened by a single database process. Due to the current file limit problem, only 2048 executors can be launched at a time if connecting to a single database host. Fault tolerance is provided by programmatic elements within the workflow and facilitated by the DBMS. As tasks are not bound to executors, the limited

executors are able to execute well over 2048 tasks successfully: while some executor connections are rejected due to file limits resulting in failure to initialize the executor process, remaining tasks are taken up by executors after previous tasks are completed.

Figure 5 shows a breakdown of the timelines for components of single-workload jobs with 60, 300, 1800 and 4200 tasks. Each task contained an executable in CUDA/C and had a duration of about 60 seconds. Timestamps were taken at the start and end of each component of the workflow by each executor. Due to short duration of the tasks, executors were able to finish tasks before all executors were dispatched and start additional queued tasks. Under 1000 executors completed the 1800 task workflow, as all 1800 tasks were running or complete with 800 executors yet to dispatch. In the 4200 task workload, connections from executors numbered 2000 and above were rejected by the database host due to the above mentioned `ulimit` settings. All tasks finished successfully.

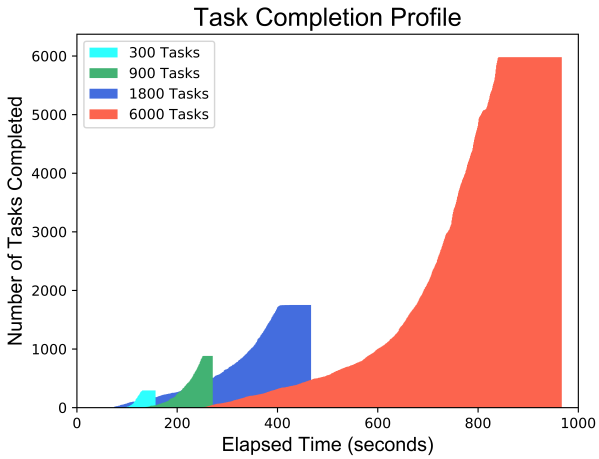

Fig. 6. Task completion profiles of workflows for 300, 900, 1800 and 6000 tasks.

Figure 6 shows task completion timing profiles for workflows with 300, 900, 1800 and 6000 total tasks. These plots show that all tasks in each case were completed the rate at which this was accomplished for each workflow instance. 900 tasks are completed faster in the 900-task workflow job than are completed within the 1800 and 6000-task jobs, we found this is due to the `ulimit` settings that caused the database to close some existing executor connections as new connection requests were handled. This resulted in a latency as the running tasks had to start again in new executors due to this executor connection failure. Figure 7, left panel, shows the task dispatch rate for varying task numbers using the CUDA executable. This rate is calculated before any task rejections occur. The maximum task dispatch rate of 19.4 tasks/sec was achieved for 4200 tasks, and a rate of about 18 tasks/sec is maintained from 2400 to 6000 tasks. A similar rate is achieved by other ensemble simulation workflow platforms [37]. This rate

is much lower than that achieved by Swift/T, but for the type of workflows targeted here, where tasks will run for hours to days, this latency is negligible compared to the total runtime of the workflow. If rapid dispatch and deployment of thousands of short simultaneous tasks is required as part of a workflow, it is possible that an instance of Swift/T can be executed inside of the workflow. Figure 7, right panel, shows the changes in the ratio of total workflow time to task number as it varies over job size. The ratio stays between 0.3 and 0.1 for all jobs with over 1000 tasks. This shows that the overhead is stable for large job sizes, and that the cost-per-task stabilizes for these large jobs. This overhead will be negligible for long jobs such as are common in MD simulations.

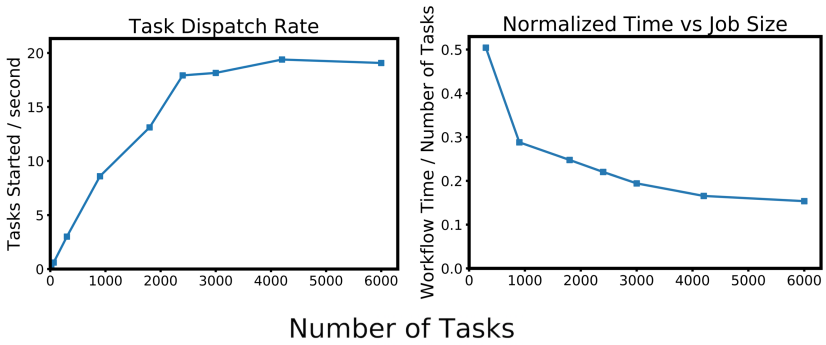

Fig. 7. Left panel: Task dispatch rate over number of tasks. Right panel: change in workflow time to task ratio over number of tasks in worflow job.

4 Conclusions and Future Work

We have shown that the use of a DBMS can provide the infrastructure for a fault-tolerant, scalable, adaptable workflow generation program that can be used to implement ESMD methods such as adaptive sampling with Markov models. Despite limitations imposed by system-level problems that must be corrected by the vendor and system administration personnel, on the Summit supercomputer preliminary tests of the components as well as initial tests of the complete program show that a workflow solution that uses Python and the MongoDB DBMS to create a dataflow-based platform for this type of workflow can be effective. Future work will involve implementation of the complete adaptive Markov-model workflow using several MD simulation programs, and investigation of possibilities for reducing database traffic using different database configurations.

5 Acknowledgements

The authors would like to acknowledge Micholas Dean Smith for help with MD test systems preparation, and Shantenu Jha and lab for extensive help with incorporation of the Radical Cybertools software stack. An award of computer time was provided by the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User

Facility supported under Contract DE-AC05-00OR227525. ORNL is managed by UT-Battelle, LLC for the US Department of Energy. FN acknowledges European Commission (ERC CoG 772230) and Deutsche Forschungsgemeinschaft (NO 825/3-1). JCS acknowledges DOE contract ERKP752. CC acknowledges support from the National Science Foundation (CHE-1265929, CHE-1740990, CHE-1900374, and PHY-1427654) and the Welch Foundation (C-1570).

References

1. Abraham, M.J., Murtola, T., Schulz, R., Páll, S., Smith, J.C., Hess, B., Lindahl, E.: GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX **1**, 19–25 (2015)
2. Adorf, C.S., Dodd, P.M., Ramasubramani, V., Glotzer, S.C.: Simple data and workflow management with the signac framework. Computational Materials Science **146**, 220–229 (2018)
3. Ailamaki, A., Ioannidis, Y.E., Livny, M.: Scientific workflow management by database management. In: Proceedings. Tenth International Conference on Scientific and Statistical Database Management (Cat. No. 98TB100243). pp. 190–199. IEEE (1998)
4. Amaro, R.E., Baudry, J., Chodera, J., Demir, Ö., McCammon, J.A., Miao, Y., Smith, J.C.: Ensemble docking in drug discovery. Biophysical Journal (2018)
5. Balasubramanian, V., Jensen, T., Turilli, M., Kasson, P., Shirts, M., Jha, S.: Implementing adaptive ensemble biomolecular applications at scale. arXiv preprint arXiv:1804.04736 (2018)
6. Balasubramanian, V., Turilli, M., Hu, W., Lefebvre, M., Lei, W., Modrak, R., Cervone, G., Tromp, J., Jha, S.: Harnessing the power of many: Extensible toolkit for scalable ensemble applications. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 536–545. IEEE (2018)
7. Balasubramanian, V., Bethune, I., Shkurti, A., Breitmoser, E., Hruska, E., Clementi, C., Laughton, C., Jha, S.: Extasy: Scalable and flexible coupling of MD simulations and advanced sampling techniques. In: 2016 IEEE 12th International Conference on e-Science. pp. 361–370. IEEE (2016)
8. Balasubramanian, V., Treikalis, A., Weidner, O., Jha, S.: Ensemble toolkit: Scalable and flexible execution of ensembles of tasks. In: 2016 45th International Conference on Parallel Processing (ICPP). pp. 458–463. IEEE (2016)
9. Bernardi, R.C., Melo, M.C., Schulten, K.: Enhanced sampling techniques in molecular dynamics simulations of biological systems. Biochimica et Biophysica Acta **1850**(5), 872–877 (2015)
10. Bowman, G.R., Pande, V.S., Noé, F.: An introduction to Markov state models and their application to long timescale molecular simulation, vol. 797. Springer Science & Business Media (2013)
11. Buchete, N.V., Hummer, G.: Peptide folding kinetics from replica exchange molecular dynamics. Physical Review E **77**(3), 030902 (2008)
12. Davidson, S.B., Freire, J.: Provenance and scientific workflows: challenges and opportunities. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 1345–1350. ACM (2008)
13. Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani, R., Chen, W., Da Silva, R.F., Livny, M., et al.: Pegasus, a workflow management system for science automation. Future Generation Computer Systems **46**, 17–35 (2015)

14. Deelman, E., Vahi, K., Rynge, M., Juve, G., Mayani, R., da Silva, R.F.: Pegasus in the cloud: Science automation through workflow technologies. IEEE Internet Computing **20**(1), 70–76 (2016)
15. Dorier, M., Wozniak, J.M., Ross, R.: Supporting task-level fault-tolerance in HPC workflows by launching MPI jobs inside MPI jobs. In: Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science. p. 5. ACM (2017)
16. Dou, L., Zinn, D., McPhillips, T., Köhler, S., Riddle, S., Bowers, S., Ludäscher, B.: Scientific workflow design 2.0: Demonstrating streaming data collections in Kepler. In: 2011 IEEE 27th International Conference on Data Engineering. pp. 1296–1299. IEEE (2011)
17. Eastman, P., Swails, J., Chodera, J.D., McGibbon, R.T., Zhao, Y., Beauchamp, K.A., Wang, L.P., Simmonett, A.C., Harrigan, M.P., Stern, C.D., et al.: OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. PLoS computational biology **13**(7), e1005659 (2017)
18. Garcia, A.E., Herce, H., Paschek, D.: Simulations of temperature and pressure unfolding of peptides and proteins with replica exchange molecular dynamics. Annual Reports in Computational Chemistry **2**, 83–95 (2006)
19. HACCmk: `https://asc.llnl.gov/CORAL-benchmarks/Summaries/HACCmk_Summary_v1.0.pdf`
20. Hänggi, P., Talkner, P., Borkovec, M.: Reaction-rate theory: fifty years after Kramers. Reviews of Modern Physics **62**(2), 251 (1990)
21. Hruska, E., Abella, J.R., Nüske, F., Kavraki, L.E., Clementi, C.: Quantitative comparison of adaptive sampling methods for protein dynamics. The Journal of Chemical Physics **149**(24), 244119 (2018)
22. Hummer, G.: Position-dependent diffusion coefficients and free energies from Bayesian analysis of equilibrium and replica molecular dynamics simulations. New Journal of Physics **7**(1), 34 (2005)
23. Husic, B.E., McGibbon, R.T., Sultan, M.M., Pande, V.S.: Optimized parameter selection reveals trends in Markov state models for protein folding. The Journal of Chemical Physics **145**(19), 194103 (2016)
24. Jain, A., Ong, S.P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G.M., Hautier, G., et al.: FireWorks: a dynamic workflow system designed for high-throughput applications. Concurrency and Computation: Practice and Experience **27**(17), 5037–5059 (2015)
25. Kasson, P.M., Jha, S.: Adaptive ensemble simulations of biomolecules. Current Opinion in Structural Biology **52**, 87–94 (2018)
26. Kubo, R.: The fluctuation-dissipation theorem. Reports on Progress in Physics **29**(1), 255 (1966)
27. Kumar, S., Rosenberg, J.M., Bouzida, D., Swendsen, R.H., Kollman, P.A.: The weighted histogram analysis method for free-energy calculations on biomolecules. I. The method. Journal of Computational Chemistry **13**(8), 1011–1021 (1992)
28. Laney, D.: Workflow project overview. `https://www.csm.ornl.gov/SOS20/documents/Laney-Workflow-Overview-SOS16.pdf`
29. Li, Y., Manoharan, S.: A performance comparison of SQL and NoSQL databases. In: 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM). pp. 15–19. IEEE (2013)
30. MacLean, C.: Python usage metrics on blue waters. Cray User Group (2017)
31. Merzky, A., Santcroos, M., Turilli, M., Jha, S.: RADICAL-Pilot: Scalable execution of heterogeneous and dynamic workloads on supercomputers. Computer Research Repository (CoRR), abs/1512.08194 (2015)

32. Merzky, A., Turilli, M., Maldonado, M., Jha, S.: Design and performance characterization of radical-pilot on Titan. arXiv preprint arXiv:1801.01843 (2018)
33. Messina, P.: The exascale computing project. Computing in Science & Engineering **19**(3), 63–67 (2017)
34. Parker, Z., Poe, S., Vrbsky, S.V.: Comparing NoSQL mongodb to an SQL db. In: Proceedings of the 51st ACM Southeast Conference. p. 5. ACM (2013)
35. Pennycook, S.J., Sewall, J.D., Lee, V.: A metric for performance portability. arXiv preprint arXiv:1611.07409 (2016)
36. Pérez-Hernández, G., Paul, F., Giorgino, T., De Fabritiis, G., Noé, F.: Identification of slow molecular order parameters for Markov model construction. The Journal of Chemical Physics **139**(1), 07B604_1 (2013)
37. Pouya, I., Pronk, S., Lundborg, M., Lindahl, E.: Copernicus, a hybrid dataflow and peer-to-peer scientific computing platform for efficient large-scale ensemble sampling. Future Generation Computer Systems **71**, 18–31 (2017)
38. Prinz, J.H.: `https://github.com/markovmodel/adaptivemd/commit/186ffa097059168cb6b17dfd2f0b01f83bc7b6e1`, git commit
39. Prinz, J.H.: `https://github.com/jhprinz`
40. Prinz, J.H., Wu, H., Sarich, M., Keller, B., Senne, M., Held, M., Chodera, J.D., Schütte, C., Noé, F.: Markov models of molecular kinetics: Generation and validation. The Journal of Chemical Physics **134**(17), 174105 (2011)
41. Pronk, S., Pouya, I., Lundborg, M., Rotskoff, G., Wesén, B., Kasson, P.M., Lindahl, E.: Molecular simulation workflows as parallel algorithms: The execution engine of Copernicus, a distributed high-performance computing platform. Journal of Chemical Theory and Computation **11**(6), 2600–2608 (2015)
42. PyMongo: `https://github.com/mongodb/mongo-python-driver`
43. Scherer, M.K., Trendelkamp-Schroer, B., Paul, Fabian an Pérez-Hernández, G., Hoffmann, M., Plattner, N., Wehmeyer, C., Prinz, J.H., Noé, F.: PyEMMA 2: A software package for estimation, validation, and analysis of Markov models. Journal of Chemical Theory and Computation **11**(11), 5525–5542 (2015)
44. Sedova, A., Eblen, J.D., Budiardja, R., Tharrington, A., Smith, J.C.: 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). pp. 1–13. IEEE (2018)
45. Sedova, A., Tillack, A.F., Tharrington, A.: Using compiler directives for performance portability in scientific computing: Kernels from molecular simulation. In: International Workshop on Accelerator Programming Using Directives. pp. 22–47. Springer (2018)
46. Sitalakshmi Venkatraman, K.F., Kaspi, S., Venkatraman, R.: SQL versus NoSQL movement with big data analytics. IJ Information Technology and Computer Science pp. 59–66 (2016)
47. Sorin, E.J., Pande, V.S.: Exploring the helix-coil transition via all-atom equilibrium ensemble simulations. Biophysical Journal **88**(4), 2472–2493 (2005)
48. Souza, R., Mattoso, M.: Provenance of dynamic adaptations in user-steered dataflows. In: International Provenance and Annotation Workshop. pp. 16–29. Springer (2018)
49. Souza, R., Silva, V., Oliveira, D., Valduriez, P., Lima, A.A., Mattoso, M.: Parallel execution of workflows driven by a distributed database management system. In: ACM/IEEE Conference on Supercomputing, Poster (2015)

50. Swenson, D.W., Prinz, J.H., Noe, F., Chodera, J.D., Bolhuis, P.G.: OpenPathSampling: A Python framework for path sampling simulations. 1. Basics. Journal of Chemical Theory and Computation (2018)
51. Trott, C.R., Plimpton, S.J., Thompson, A.P.: Solving the performance portability issue with Kokkos. (2017)
52. Turilli, M., Santcroos, M., Jha, S.: A comprehensive perspective on pilot-job systems. ACM Computing Surveys (CSUR) **51**(2), 43 (2018)
53. Weinan, E., Ren, W., Vanden-Eijnden, E.: String method for the study of rare events. Physical Review B **66**(5), 052301 (2002)
54. Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., et al.: The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Research **41**(W1), W557–W561 (2013)
55. Woolf, T.B., Roux, B.: Conformational flexibility of o-phosphorylcholine and o-phosphorylethanolamine: a molecular dynamics study of solvation effects. Journal of the American Chemical Society **116**(13), 5916–5926 (1994)
56. Wozniak, J.M., Armstrong, T.G., Wilde, M., Katz, D.S., Lusk, E., Foster, I.T.: Swift/T: Large-scale application composition via distributed-memory dataflow processing. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. pp. 95–102. IEEE (2013)
57. Wu, H., Paul, F., Wehmeyer, C., Noé, F.: Multiensemble Markov models of molecular thermodynamics and kinetics. Proceedings of the National Academy of Sciences p. 201525092 (2016)