Generalizable Coordination of Large Multiscale Workflows: Challenges and Learnings at Scale

Harsh Bhatia[*]
hbhatia@llnl.gov

Francesco Di Natale[†]
dinatale3@llnl.gov

Joseph Y. Moon[*]
moon15@llnl.gov

Xiaohua Zhang[‡]
zhang30@llnl.gov

Joseph R. Chavez[§]
chavez35@llnl.gov

Fikret Aydin[‡]
aydin1@llnl.gov

Chris Stanley[¶]
stanleycb@ornl.gov

Tomas Oppelstrup[‡]
oppelstrup2@llnl.gov

Chris Neale[‖]
cneale@lanl.gov

Sara Kokkila Schumacher[**]
saraks@ibm.com

Dong H. Ahn[*]
ahn1@llnl.gov

Stephen Herbein[*]
herbein1@llnl.gov

Timothy S. Carpenter[‡]
carpenter36@llnl.gov

Sandrasegaram Gnanakaran[‖]
gnana@lanl.gov

Peer-Timo Bremer[*]
bremer5@llnl.gov

James N. Glosli[‡]
glosli1@llnl.gov

Felice C. Lightstone[‡]
lightstone1@llnl.gov

Helgi I. Ingólfsson[‡]
ingolfsson1@llnl.gov

ABSTRACT

The advancement of machine learning techniques and the heterogeneous architectures of most current supercomputers are propelling the demand for large multiscale simulations that can automatically and autonomously couple diverse components and map them to relevant resources to solve complex problems at multiple scales. Nevertheless, despite the recent progress in workflow technologies, current capabilities are limited to coupling two scales. In the first-ever demonstration of using *three scales of resolution*, we present a scalable and generalizable framework that couples pairs of models using machine learning and *in situ* feedback. We expand upon the massively parallel Multiscale Machine-Learned Modeling Infrastructure (MuMMI), a recent, award-winning workflow, and generalize the framework beyond its original design. We discuss the challenges and learnings in executing a massive multiscale simulation campaign that utilized over 600,000 node hours on *Summit* and achieved more than 98% GPU occupancy for more than 83% of the time. We present innovations to enable several orders of magnitude scaling, including simultaneously coordinating 24,000 jobs, and managing several TBs of new data per day and over a billion files in total. Finally, we describe the generalizability of our framework and, with an upcoming open-source release, discuss how the presented framework may be used for new applications.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Multiscale systems**; **Massively parallel and high-performance simulations**; *Simulation tools*; • **Applied computing** → **Computational biology**.

KEYWORDS

multiscale simulations, adaptive simulations, massively parallel, heterogenous architecture, machine learning, cancer research

ACM Reference Format:
Harsh Bhatia, Francesco Di Natale, Joseph Y. Moon, Xiaohua Zhang, Joseph R. Chavez, Fikret Aydin, Chris Stanley, Tomas Oppelstrup, Chris Neale, Sara Kokkila Schumacher, Dong H. Ahn, Stephen Herbein, Timothy S. Carpenter, Sandrasegaram Gnanakaran, Peer-Timo Bremer, James N. Glosli, Felice C. Lightstone, and Helgi I. Ingólfsson. 2021. Generalizable Coordination of Large Multiscale Workflows: Challenges and Learnings at Scale. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3458817.3476210

[*]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, California, 94550
[†]Applications, Simulations, and Quality, Lawrence Livermore National Laboratory, Livermore, California, 94550
[‡]Physical and Life Sciences, Lawrence Livermore National Laboratory, Livermore, California, 94550
[§]Global Security Computing Division, Lawrence Livermore National Laboratory, Livermore, California, 94550
[¶]Computational Sciences and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 37831
[‖]Theoretical Biology and Biophysics, Los Alamos National Laboratory, Los Alamos, New Mexico, 87545
[**]IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 10598

1 INTRODUCTION

Computational solutions to complex problems in all spheres of science and engineering rely increasingly on multiscale modeling and simulations to capture intricate phenomena at longer and larger time- and length-scales than previously possible [15, 19, 32, 37, 45]. As a result, recent years have seen a remarkable shift away from the realm of large monolithic simulations [30, 33, 69] toward massive parallel ensembles [17, 24, 38, 58, 61] consisting of thousands of smaller simulations at different scales, orchestrated through sophisticated workflows. The thrust toward Exascale computing [25] and the emerging supercomputing technologies — particularly heterogeneous architectures — further underscore the need for large, cross-functional workflow technologies comprising orchestrators, services, simulations, machine learning, data analytics, and visualization to the extent that workflows are now being called the new high-performance computing (HPC) applications at Exascale [9].

Enabling such large multiscale simulation campaigns on next-generation machines requires a workflow infrastructure capable of connecting multiple, diverse components easily, dynamically, and transparently, while balancing the load in the computational effort and managing the resulting data. While no general solutions exist, efforts are underway to make these goals a reality.

In particular, the recent presentation of MuMMI [37] offers a new paradigm for large multiscale simulations that couples two scales — a *macro* and a *micro* — using a dynamic sampling driven by machine learning (ML) [12] and *in situ* feedback. The MuMMI workflow [24] is capable of orchestrating such massive multiscale campaigns at unprecedented computational scales, with effective utilization of *all* of the heterogeneous resources on some of the largest machines on the planet (demonstrated on *Sierra*, using 176,000 CPU cores and 16,000 GPUs). This workflow can coordinate several thousand simultaneous jobs, administer diverse components, and generate and manage multiple TBs of data. Nevertheless, despite its initial success, several limitations in MuMMI prevented generalization to other scientific inquiries and scaling up the infrastructure.

In this work, we make two key technological advances. First, we *generalize* the scope and design of the MuMMI framework to facilitate broader applicability. Toward this goal, we present a more-advanced workflow management that uses generic strategies to couple additional modeling techniques and computational methodologies, requiring diverse software packages and needing versatile types of jobs, data, and compute, as well as support for database technologies of choice. In particular, we demonstrate *three resolution scales* to investigate a new scientific phenomenon of interest and incorporate the necessary specific details, including two levels of ML and two types of feedback. With a pending open-source release of MuMMI, we also present guidelines to customize and further extend this framework to support other scientific studies.

Second, we tackle several computational bottlenecks within the MuMMI workflow and, using new strategies, demonstrate significantly improved *scaling* in performance. Examples of such limitations were a slow feedback mechanism and bundled-scheduling approach for simulations. Our technical innovations improve not only the scalabilty of the MuMMI framework but also its generalizability by allowing a more elastic coupling of scales that may further broaden its applicability.

Contributions. This paper reports on the novelty that makes this framework sufficiently *generalizable* to incorporate three resolution scales (and potentially other applications) and support different software tools in various HPC environments and sufficiently *scalable* to enable an even larger scientific campaign. We describe the associated computational challenges and the strategies to tackle the seemingly mundane idiosyncrasies of modern supercomputers. Specifically, we make the following contributions.

(1) *Generalizations* of MuMMI: (a) support for diverse modeling techniques, with a specific focus on an extension to **three resolution scales**, two layers of ML-based selection, and two types of feedback; (b) incorporation of database technologies; and (c) generic, extendable APIs and open-source framework.
(2) *Scaling* of the workflow performance: (a) more than **12× faster feedback** mechanism to enable more-frequent coupling of scales; (b) explicit simulation-to-job mapping to directly control each simulation, supported by an almost **3× faster job scheduling**; and (c) more-efficient ML framework supporting almost **165× more data** for dynamic, real-time decision making.
(3) A *demonstration* of the largest simulation campaign of its kind: (a) including a **full-system run** on *Summit*, the second-most-powerful supercomputer in the world, (b) utilizing **over 600,000 node hours**, (c) with **more than 98% GPU occupancy** for more than 83% of the time, (d) simultaneously coordinating **24,000 jobs**, (e) creating and managing **several TB's of data each day,** and (f) handling over **a billion files** in total.

2 RAS-RAF-MEMBRANE DYNAMICS

Mutations of RAS proteins are implicated in nearly a third of all human cancers diagnosed in the US, including significant proportions of pancreatic (∽95%), colorectal (∽45%), and lung (∽35%) cancers [59, 66]. Consequently, there is significant interest in unraveling the underlying biological mechanisms in order to develop effective treatments. However, despite studying RAS and, in particular, KRAS (the most frequently mutated form of RAS [71]) in various cancers for years, no FDA approved therapies that target these mutations are currently available; thus, RAS has often been labeled *undruggable* [43]. Nevertheless, recent progress [53] shows that fundamental research encourages therapeutic development.

RAS proteins are localized on the inside of cellular plasma membranes (PMs) and act as molecular switches in cell growth signaling. Only activated RAS proteins can initiate signaling by binding to downstream effector proteins, particularly, RAF proteins. The mechanism by which RAS and RAF localize on the PM is hypothesized to be crucial for activating the signaling pathway. However, the precise role of the membrane composition (*e.g.,* charged vs. neutral lipids), membrane dynamics (*e.g.,* undulations and domain formation), and other physiochemical properties in affecting such localization is not fully understood.

Computational modeling of these phenomena is challenging because the binding of RAS and RAF at the PM is inherently

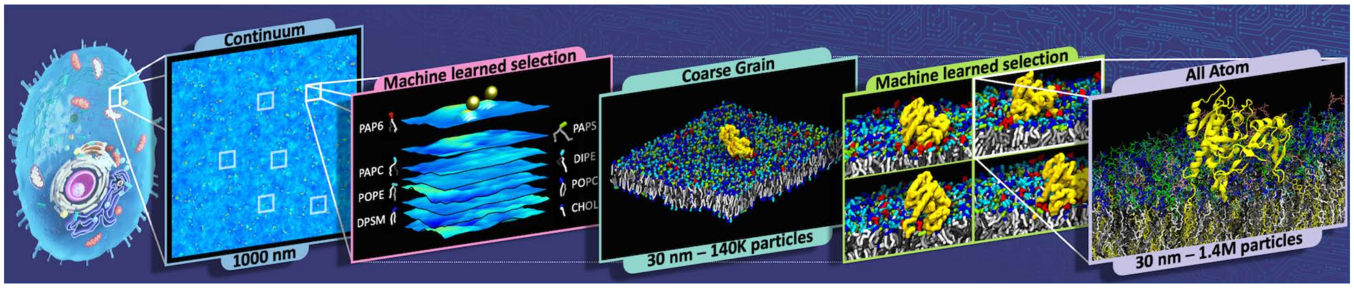

Figure 1: We showcase a massive simulation campaign that focuses on an important signaling mechanism to the cell membrane through RAS-RAF protein complex and simulates the relevant interactions at *three scales of resolution.* These scales are coupled through ML-based selection in a pairwise manner to promote important configurations from coarse to fine scale.

a multiscale process. Particle-based models are inefficient at exploring all unique molecular interfaces between RAS and RAF and incur long wait times for speculative association, which are largely dependent on diffusion. Different types and scales of molecular dynamics (MD) simulations are used to model the relationships between structure, dynamics, and function in biological macromolecules. In particular, *all-atom* (AA) [44, 56, 64], *coarse-grained* (CG) [31, 36, 51, 63], and ultra-CG [55, 60, 73] models have been used to simulate biomolecular systems. Long continuous simulations have extended into the high μs to ms range [35, 56, 64, 71], and large ensembles of shorter simulations have accumulated multiple ms [6, 44, 52, 72, 76]. Nevertheless, a typical limitation of such simulations is that the ones that are expansive in either size or duration are inadequate in the other dimension, *i.e.,* large but short or long but small.

This is precisely the challenge addressed by the multiscale modeling infrastructure MuMMI [24, 37]. To expand both the time- and length-scales concurrently, MuMMI uses relatively inexpensive models for rapid exploration coupled with more-descriptive models that provide sufficient resolution. For example, a macro model using a continuum description to evolve lipids according to dynamic density functional theory (DDFT) [50] can easily achieve ms and μm scales [37], which MuMMI uses to spawn and direct MD simulations at the required fidelity. Here, we expand upon MuMMI to characterize the key events that trigger oncogenic signaling in RAS-RAF-membrane systems, as depicted in Figure 1.

3 RELATED WORK

Commonly available batch systems such as SLURM [77] and LSF® [67] rely on the ability to utilize MPI-type communication to span a set of resources with fixed resource requirements. However, with a significant departure from the traditional approach of large, single application jobs, modern workflows are becoming increasingly complex webs of intercommunicating stages and microservices [9].

There exist several solutions focusing on assembling complex post-processing capabilities [5, 22, 39] and large static ensembles [21, 27, 57] as well as tools for orchestrating such parameter studies [2, 8, 23]. Broadly, such tools lack the flexibility needed for dynamic workflows. In contrast, there also exist solutions [16, 20, 26] that can couple different solver codes together by exchanging information, such as boundary conditions or even entire grids, between the solvers. However, such approaches, by design, provide extreme tight coupling, are intrusive, and cannot be adapted easily.

Recently, there has been a growing focus on the use of ML in workflows to dynamically steer the ensemble towards configurations of import and to overcome the limits of computational scales otherwise achievable. MuMMI [24, 37], a recent, award-winning framework, is a premier example of ML-driven, dynamic workflows and offers a new paradigm of multiscale simulations by coupling two scales of resolution, demonstrated in the context of cancer research. The more-recent works of Casalino *et al.* [17] and Jacobs *et al.* [38] also utilize different forms of deep learning to deliver large multiscale simulations in exploration of COVID-19.

MuMMI: Multiscale Machine-Learned Modeling Infrastructure. Of special relevance to our work is MuMMI [24, 37], which provides a bidirectional coupling of two scales — a *macro* and a *micro* — using ML for forward coupling and *in situ* feedback for backward. By using ML to dynamically select the most novel macro configurations [12], MuMMI continuously steers the multiscale simulation towards new exploration and, given enough time, can simulate every type of configuration either directly or as a proxy to a similar enough configuration. MuMMI also analyzes the ongoing micro scale simulations and can use their results to update the less-accurate, macro model, thereby, creating a self-healing mechanism, which, given enough time, will improve the accuracy of the coarser model. As a result, MuMMI can achieve macro time- and length-scale, but with an effective accuracy of micro scale.

With a DDFT-based continuum model as the "macro scale" and a CG model as the "micro scale", MuMMI previously coupled two models to study the interactions of RAS proteins with the PM — a simpler and smaller version of our target study, which makes MuMMI a potential solution for our goals. MuMMI was demonstrated to scale on all of *Sierra* [24], occupying all of the available GPUs to create a massive ensemble of CG simulations, which led to new insights into RAS protein dynamics on the PM and the influence of lipids and lipid fingerprints [37].

Despite its capabilities and successful demonstration, MuMMI was tailored to the two specific models used previously [24, 37] and had certain computational limitations that prevented further scaling. Among others, MuMMI was completely reliant on the filesystem and, therefore, was bottlenecked by the GPFS throughput. This limitation resulted in a need to explicitly throttle the rate of certain I/O operations, most prominently, during feedback. Furthermore, scaling in job scheduling was obtained by bundling jobs of similar kind to alleviate the load on the scheduler. Nevertheless, such an approach is undesirable since it prevents explicit control over individual jobs. Broadly, all components of MuMMI, including job and data management, data and control communication, as well as the exposed API, were tightly integrated with the specific problem, preventing utilizing MuMMI for our new application.

4 GENERALIZABLE AND SCALABLE MUMMI

In this work, we present a new design that expands the generalizability and scalability of MuMMI and demonstrates these innovations by extending the MuMMI workflow to support a third scale of resolution. Hereafter, unless explicitly noted, MuMMI refers to our new, improved, and generalizable framework.

Broadly, a multiscale model can be developed through pairwise coupling of scales. For any two scales, some basic building blocks are needed: (1) simulation and analysis at the two (coarse/fine or macro/micro) scales, (2) a method to couple the two representations, (3) an automated approach to decide which coarse representations to promote to the fine scale, and (4) a method to perform feedback.

In this context, we design MuMMI as comprising two parts — the *application* and the *coordination* (see Figure 2). The former defines the application scope (in terms of the building blocks listed above), *e.g.,* what scales are relevant, what codes and/or simulation tools to use, what ML techniques are suitable, and how is the feedback performed? These components are typically designed by computational scientists who are experts in the corresponding domains; the actual details may vary across applications or even across simulations. The role of the generalized MuMMI workflow (the coordination part) is to tie together the different application components to facilitate the multiscale simulations.

We first discuss the specific details of our three-scale application, followed by generic and tailored strategies for coordination.

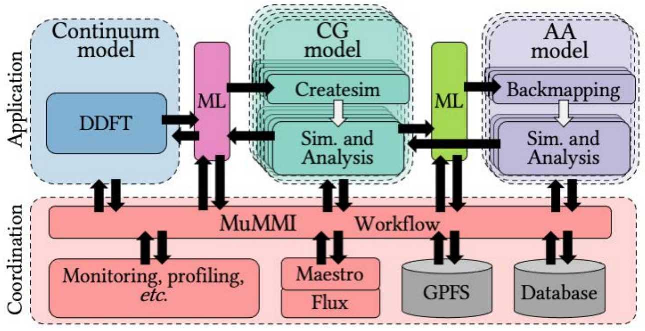

Figure 2: We present a generalizable and scalable framework to couple diverse models at different resolution scales. The "application" components (top) define the three scales and may be swapped to support other applications, whereas the "coordination" components (bottom) provide an interface to couple the associated tools, software components, and technologies to facilitate scalable simulation campaigns.

4.1 The Three-Scale MuMMI

This work uses three scales of resolution: *continuum*, *coarse grained* (CG), and *all atomistic* (AA), along with two types of ML-based selection and two types of *in situ* feedback. Although every application component used in this work has notable innovations in itself, whether modeling, development, or performance, we describe these components only briefly, focusing largely on their considerable computational versatility that challenges the workflow.

(1) The Continuum Simulation. The coarsest of the three scales is a *macro model* that provides speed at the cost of accuracy. Our macro model is a *continuum* description of lipids that uses DDFT [50] for representing lipid dynamics in terms of their density fields. Proteins (positions and configurational states) are represented as particles that interact with each other and with the lipids This model comprises a 1 μm × 1 μm bilayer discretized as a 2400×2400 grid, with 8 lipid types in the inner and 6 types in the outer leaflet [34]. We use a custom simulation package, *GridSim2D* — a parallel CPU code written in C++ that uses MPI for communication. Using a total of 3600 MPI ranks (24 CPU cores per node at 150 nodes), *GridSim2D* can simulate ∽0.96 ms per day of walltime. With an I/O rate of 1 μs, a new snapshot is delivered every 90 seconds and, when stored in a custom binary format, consumes ∽374 MB of disk space.

(2) Createsim: Mapping Continuum-to-CG. Compared to the continuum scale, the CG and AA simulations are restricted in the spatial extent due to high computational cost. To couple continuum with these scales, 30 nm × 30 nm *"patches"* are cut out of continuum snapshots in regions that may be of interest for CG and AA simulations. The *createsim* module transforms a patch from continuum representation into a particle-based one. The *insane* tool [74] is used to create a CG representation of the membrane and proteins. Once constructed, GROMACS [1] is used to relax the membrane and proteins into a more natural, equilibrated, state in preparation for simulation. *Createsim* is a custom Python-based code that uses 24 CPU cores and, on average, takes ∽1.5 hours to complete.

(3) CG Simulations and Analyses. Given the particle representation of lipids and proteins, CG simulations with the Martini force field [51] are performed using the CUDA®-enabled version [78] of ddcMD [68]. Custom, Python-based analysis is executed simultaneously on the same computational node and accesses the local on-node RAM disk for analysis of the MD trajectories generated by the corresponding simulation. Each ddcMD simulation uses one GPU and one CPU core; the corresponding analysis is allocated 3 CPU cores. With this setup and an average of ∽140,000 particles, ddcMD delivers ∽1.04 μs of MD trajectories per day per GPU [78], and produces about 4.6 MB new data every 41.5 seconds. The analysis module is tuned to finish inspecting each snapshot within this time period and generates 17 KB additional data every 41.5 seconds.

(4) Backmapping: Mapping CG-to-AA. To overcome the limitations of the CG model [4], it is further refined using a *backmapping* scheme that translates a CG representation in time into AA using the CHARMM36 force field [10]. This procedure retrieves a selected snapshot from the ddcMD trajectory, converts the CG to the AA model using a modified version of the *backward* tool [75], performs cycles of energy minimization and position-restrained MD using GROMACS [1], and finally converts the data format from GROMACS to AMBER using ParmEd [65]. Using 18 CPU cores, each backmapping run takes ∽2 hours on average to complete. Each backmapping calculation produces 2.9 GB data every 2 hours on the local on-node RAM disk and about 0.5 GB data is backed up to GPFS to initialize an AA simulation.

(5) AA Simulations and Analyses. Once backmapped, the AA configurations are simulated using the AMBER MD simulation package [18, 62]. One GPU is allocated to each simulation because the multi-GPU setup is inefficient due to slow communication across GPUs or across nodes. Instead, similar to the CG case, many single-GPU MD simulations are run in parallel to achieve better GPU utilization, and a Python-based analysis module is executed that analyzes new AA trajectory snapshots as soon as they are generated. The average atomistic simulation system includes 1.575 M atoms and the simulations generate almost 13.98 ns per day per GPU. A simulation produces one frame every 10.3 minutes at 0.1 ns frame rate, where the size of each frame is about 18 MB.

(6) ML-based Patch and CG Frame Selection. Our three-scale simulation campaign requires two layers of sampling to couple the scales in a pairwise manner — both layers are similar in philosophy but differ in technical details. The 30 nm × 30 nm patches extracted from the continuum data are evaluated for "novelty" in a reduced, 9-D representation generated by a metric learning approach implemented using a deep neural network. Similar to the work of Bhatia *et al.* [12], a farthest-point sampling approach is used to identify novel configurations, although our patches are almost 55× larger (sampled on a 37×37 grid instead of 5×5). In the case of relevant CG frames, the conformational state of the RAS-RAF complex is coded using a 3-D representation, which unlike the representation of patches is not conducive to farthest-point sampling. As a result, a new framework is developed to identify novel frames through a discrete, histogram-based sampling in this 3-D space.

(7) CG-to-Continuum and AA-to-CG Feedback. The CG-to-Continuum feedback aggregates the protein-lipid radial distribution functions (RDFs) computed through the online analysis of CG simulations and propagates the aggregated result to the ongoing continuum simulation, which reads and updates these parameters on the fly. Each feedback iteration must rapidly process new frames (specifically RDFs), which are expected to be created every 3–4 minutes per simulation, or 900–1200 new frames per minute for a moderately sized allocation that runs 3600 CG simulations simultaneously. In the case of AA-to-CG feedback, the secondary structures of the proteins are calculated from AA frames and analyzed to determine the most common pattern of protein secondary structure observed in the AA simulations. The force field parameters of the CG protein model depend on the secondary structure, and, therefore, the parameters are progressively refined to enable a more accurate CG model. Although the rate of incoming frames is lower for this type of feedback, each frame requires longer processing: 2400 new frames every ∽10 minutes (assuming 2400 AA simulations at 1000-node scale), and processing each frame needs two system calls to an external module, taking ∽2 s in isolation.

4.2 Generic Framework for Data Management

A natural by-product of large simulation campaigns enabled by MuMMI is the tremendous amounts and variety of data. The size of data is usually only one of the many concerns, as computational limitations may be related to the intended use of the data. For example, some data may need to be accessed only *after* the simulation whereas other may need to be read/updated by one or more simulation components *in situ*. Both situations pose different types of problems and require different solutions. MuMMI's earlier approach for data management was to use effective archiving of data, which works very well for the former (*i.e.,* write only) but not so much for the latter. In order to support frequent feedback loops (*i.e.,* requiring fast access and a systematic way of tracking what data has been processed), database-based approaches are ideal but they may not support very large files, *e.g.,* MD trajectories.

Rather than speculating on all possible scenarios and creating tailored implementations, we have developed an abstract notion of a *data interface* to support different specific backends. Currently, we use three backends: *filesystem*, *taridx*, and *redis*. By providing an abstract API, it becomes possible to have custom implementations of standard data formats, *e.g.,* save a Numpy archive into a byte stream that can be redirected effortlessly to a file, an archive, or a database — all with a single configuration switch. The availability of these data interfaces in MuMMI provides immense flexibility to the application developer as the specific modules can be implemented largely agnostic to the read-write details, and to the workflow developer as different specific interfaces can be implemented and tested in isolation and relatively easily, reducing the the overhead for development of new functionality and applications.

The simplest data interface accesses the filesystem directly, *i.e.,* reads/writes data from/to disk. This functionality is most suitable for small files, *e.g.,* those that store the state of the simulation (checkpoints, logs, *etc.*), those that may need to be analyzed or transferred in isolation, and those that may need to interface with standard tools (*e.g.,* GROMACS, AMBER, *etc.*) and hence are constrained to certain nonstandard data formats. Where needed, I/O armoring and redundancy is used to guard against filesystem failures, *e.g.,* backups of checkpoint files and retrials if reading/writing fails.

Nevertheless, saving individual files at scale can throttle the filesystem due to the sheer amount of inodes. One of the simplest ways of reducing the inode count is to collect files into archives. To support archiving millions of files, we provide random access through a complementary index file. We have expanded MuMMI's archiving capabilities and packaged them into a more-generic and more-scalable module, *pytaridx*, which is used for managing arbitrary data formats (*i.e.,* read/write of generic byte streams and text streams) with a fast throughput. The archives created using the *pytaridx* are standard tar files, which are portable and can be used with the commonly-available decoder. Archiving the data is a simple and elegant solution for managing a large number of files. By design, this approach prevents data corruption due to hardware/software failures since the file is written in append mode only.

The same functionality, however, prevents updating the data if needed. In many cases, data may need to be updated, deleted, or moved after certain processing steps, making the archiving strategy unsuitable. Although one may explicitly manipulate the associated index files to "remove" a key, the data itself cannot be updated. Therefore, for *in situ* feedback loop, we employ a database backend. In particular, we use the Python interface of Redis™ [46] to store any data that needs high throughput and update operations. MuMMI's *redis* interface sets up a cluster of Redis servers that are allocated randomly to all compute nodes. Any MuMMI component, application or coordination, can interact with the *redis* interface for data queries, whereas all internal details (*e.g.,* database and cluster) are abstracted from the user. By utilizing Redis, we eliminate the need to store and read RDFs from disk; instead, we leverage Redis as a short-term and highly responsive in-memory cache to reduce the amount of time per feedback loop. Furthermore, performing feedback through Redis reduces the load on GPFS and allows it to be performed away from any additional mechanisms that may severely delay read/write from/to a highly contested directory, such as directory locking and other OS level blocking calls.

4.3 Improvements in Job Scheduling

The MuMMI workflow facilitates a deliberate resource placement strategy to maximize simulation throughput and provide effective use of heterogeneous resources. There are three key novelties that we have incorporated in MuMMI's scheduling approach.

First, we assign GPUs to simulations individually rather than per-node — a crucial functionality both for explicit control over the simulations and for effective use of heterogeneous architectures. Previously [24], MuMMI scaled the job scheduling by bundling simulations on compute nodes, with each simulation in the bundle consuming one GPU (on *Sierra*, 4 GPUs/node translated to 4 simulations/job). Although scalable, this bundling strategy prevents controlling each simulation explicitly, reducing the effective use of resources (with the worst case utilization of 1/4, when a single simulation keeps the job alive and continues to occupy the node). This limitation would only exacerbate when moving to *Summit* (6 GPUs/node leads to worst case utilization of 1/6). To facilitate effective use of resources for simulations of interest and to provide direct control of the simulations to the application, we "unbundle" the jobs and place each simulation as an independent job, even at the cost of 6× increase in the number of jobs.

Second, we make explicit use of subprocesses to combine the simulation and corresponding analysis for direct user control over the simulation. This approach simplifies and generalizes the workflow in two ways: MuMMI needs to monitor only the Python-based analysis job, which internally handles the starting, monitoring, checkpointing, and, if needed, restoring a simulation, as well as ensuring that the *in situ* analysis keeps up with the ongoing simulation. Second, this allows an application developer to implement new types of simulation and analysis easily, without intrusion from the workflow. MuMMI provides an abstract framework and several utilities for defining simulations beyond our current application.

Finally, to support handling arbitrary types of jobs, we provide a generic and abstract *Job Tracker* that can be customized using a combination of inherited classes and configuration files. Without loss of generality, describe the resource and job placement (on *Summit*) for our target simulation campaign, which requires four types of jobs: CG setup, CG simulation/analysis, AA setup, and AA simulation/analysis. In this case, both types of simulations use one GPU each and are bound to two CPU cores each that share cache. Each corresponding analysis task is placed on a small number of CPU cores that are closest to the PCIe bus to ensure fast data movement. Finally, all setup jobs work exclusively on CPU cores and are assigned 24 cores within a node, reserving all GPUs for simulations only and preventing inter-node communication.

Tools for Job Scheduling. To achieve portability in job scheduling, the MuMMI workflow interfaces with Maestro [23], which provides a consistent API to schedule and monitor jobs. At the back-end, Maestro can interface with different job schedulers. By absorbing the changes and peculiarities of different job schedulers, Maestro allows MuMMI to be agnostic to the specific choice of scheduler.

For job scheduling, we use Flux [3], which is a fully-capable HPC workload manager, equipped with both a hierarchical resource manager and a batch-job scheduler, and has many features designed to meet the needs of emerging, large-scale workflows. One particularly useful feature is the *single-user mode*, which allows the user to instantiate an "isolated HPC system" within a standard batch allocation, facilitating complete control over jobs within the workflow. Flux provides many policy knobs to customize the scheduling behavior. Here, we select throughput-oriented options for queuing (*i.e.,* first come, first served with no backfilling) as well as resource matching (*i.e.,* low resource ID first) to map the different type of jobs precisely according to their resource and affinity requirements.

4.4 Workflow Management

MuMMI is coordinated by a configurable *Workflow Manager* (WM). Generically, the role of the WM is to couple the scales by consuming relevant data (in this case, from continuum and CG simulations), supporting ML-based selection, spawning the corresponding simulations (CG and AA, respectively), and facilitating a feedback loop by ingesting and aggregating data from up to tens of thousands of running simulations. The WM is also responsible for tracking all running jobs, managing data, profiling, and several other tasks. Here, we detail some key functions performed by the WM in the specific context of the RAS-RAF-PM simulation campaign.

Task 1: Process coarse-scale data for consumption. The data generated by the continuum simulation spans 1 μm × 1 μm and must be parsed to generate 30 nm × 30 nm *patches* of interest (regions around RAS and RAF proteins). The WM coordinates the *Patch Creator*, which reads each snapshot, creates patches, and outputs them for consumption by the rest of the framework. It takes ∽14 s to process a single snapshot and save the resulting patches in a standard Numpy format; each patch occupies about 70 KB of disk space and offers simple and portable I/O. Processing CG data is challenging, since unlike the continuum scale (only one simulation), several thousand CG simulations are executed simultaneously. Parsing thousands of CG trajectories to extract *frames* poses significant I/O cost on the (central) WM and is, therefore, done in a distributed manner. For fast throughput, each CG analysis outputs the frames of interest in the form of identifying information (∽850 B) that is minimal and sufficient for the downstream tasks.

Task 2: Select important patches/frames for spawning new CG/AA simulations. A custom, abstract API was developed using the DynIm framework [12, 14] that was extended by both the *Patch Selector* and the (CG) *Frame Selector* to implement specific procedures. Both selectors operate on DynIm's *high-dimensional point* objects and, hence, are agnostic to the specific encoding of patches and frames. These encoded representations may be computed using a ML inference engine (as done by the *Patch Selector*), a simpler dimensionality reduction (*e.g.,* principal component analysis), or any configurational representation (as done by the *Frame Selector*).

New candidates (patches and frames) for selection are ingested by the WM as soon as new data is generated, whereas new selections are made upon request, *i.e.,* when simulations turn over and/or new resources are available. Since selection events are orders of magnitude fewer than addition events, we use a caching scheme to postpone expensive computations until the time of a selection, which makes the cost of adding new candidates negligible. Both selectors also support dynamic and almost-real-time selection.

To support the application need, we incorporate five in-memory queues in the *Patch Selector* for sampling different protein configurations. For computational viability, each queue is capped at 35,000 patches. When fully populated, it takes 3–4 minutes to update the ranks of all candidates within all queues; then, the cost of selecting the top candidates is trivial. Given the 9-D encoding of patches, the ranks are updated using approximate nearest neighbor queries (with L2 distances) powered by the FAISS framework [41, 42].

Unlike the encoding used for patches, the *Frame Selector* relies on a 3-D encoding of CG frames that represents three disparate quantities; therefore, the L2 distance is not meaningful. To support a functionally useful sampling, a *binned sampler* was developed (using the DynIm API) that allows treating the three dimensions of the encoding separately. The binned sampling approach also facilitates control over the balance between importance and randomness — another functional requirement for the selection of CG frames. This new sampling approach is capable of providing significantly faster updates to ranking: 3–4 minutes for 9 M candidates.

Task 3: Schedule and manage (tens of thousands of) jobs. To maximize resource occupancy, the WM regularly scans all running jobs to determine completion (either success or failure) and submits new jobs (or resubmits failed ones) to re-engage resources as soon as they become available. If a new setup job (*createsim* or *backmapping*) is needed, a new selection (of patch or CG frame, respectively) is made, and the just-concluded setup job(s) are queued for the corresponding simulation. When new simulation jobs are needed, these queued ones are picked. To prevent GPU downtime, sets of CG and AA simulations are kept prepared (setup completed) in anticipation. The sizes of these sets are a trade-off between readiness for availability of resources and simulating stale configurations. This user-configurable trade-off governs the utilization of CPUs since setup jobs work on CPUs only and a full buffer prevents new setup jobs. We note that these specific details about the job monitoring can be controlled by the user through specific configurations of different types of *Job Trackers*, *e.g.,* the interdependence of jobs, the resource and time requirements, and the specific ways of assessing success and failure of the simulations.

Task 4: Facilitate frequent feedback iterations. Generically, a feedback iteration collects data from all running simulations, processes it, and reports the analysis. A new abstract API, the *Feedback Manager* was developed to allow controlling the specific details.

In the case of CG-to-Continuum feedback, whereas the rate of incoming data is high (900–1200 new frames per minute for a typical allocation), each data point itself is small and easy to process. New frames can be fetched in parallel (when reading from files) or serial (when using a high-throughput database). Next, whereas the reporting of the results is trivial, the WM needs a way to "tag" the processed frames to prevent reprocessing. Maintaining this information in-memory would be simple but prohibitive at scale, requiring comparisons of (the ids of) new frames against processed ones and checkpoint this in-memory information frequently to guard against hardware or software failures. Instead, we use an alternate strategy of moving each processed frame out of the relevant *namespace* (*i.e.,* moving files to tar archives or renaming keys in the database). Although this strategy adds to the time needed to complete any given feedback iteration, it lends immense scalability since this cost scales only with the number of ongoing simulations, and not with the total simulation frames ever generated. The AA-to-CG feedback uses similar strategies, with a key difference being that it deals with fewer frames, each of which takes longer to process. Together with customizable backends and tailored multiprocessing pools, these improvements and the abstract *Feedback Manager* allowed us to tailor both types of feedback to finish within the given time limit.

Parallelism and Locking. These four tasks are largely independent and can be performed in parallel. Nevertheless, these processes may need to share certain objects (*e.g., Patch Selector* is used for both ML selection and feedback). The WM facilitates appropriate mechanisms to prevent race conditions. Specifically in this work, thread-safe objects are used with a mix of blocking and nonblocking locks. Although essential, these strategies also impact performance, *e.g.,* when feedback process may have to wait to acquire the lock.

Resilience to System Failures. MuMMI creates its resilience strategy by composing fault tolerance support built directly into some of the key software tools it uses. For example, Flux provides resilience against compute node failures; it has full support to detect node failures and to drain the failed nodes so that no new jobs can be scheduled while keeping the existing jobs running. Similarly, Redis is an industry standard that utilizes redundancy to mitigate failures in communication. MuMMI is designed to inherit much of the basic resilience support in these tools. Furthermore, our workflow employs thorough checkpointing mechanisms to guard against software and hardware failures and can be restored completely after any such crash without much loss of data. All simulations are checkpointed with their own simulation code at ∽15 min interval, and any missing frames can be rerun, if needed. Backups of checkpoint files are also maintained to mitigate issues of filesystem failures during checkpointing. Data I/O and communication is organized in a way that if the data producer fails, the consumer components simply wait until the producer is restored and continues to provide new data. On the other hand, if a consumer fails, the unconsumed data simply aggregates and is supplied when consumer is brought up online again. In addition to checkpointing, key components (ML and job scheduling) also maintain elaborate history files that may be replayed exactly, if necessary. Our archiving strategy is also robust against failures — in the event of a failure during a write, the same key gets reinserted and is taken to be the correct value.

4.5 Portability and Generalizability

The two-part software architecture of MuMMI is highly generalizable. The first part (coordination) is composed of several general-purpose and well-tested (*e.g.,* Redis, Flux, and Maestro) or newly built (*e.g.,* pytaridx and DynIm) tools to provide a completely general workflow and data management platform. Through its second part, MuMMI plugs domain-specific logic into this platform. Most of such composition is done via well-defined interfaces. Whereas some of the highly domain-specific components may not be general, the overall framework itself remains highly general as other applications can swap out our domain-specific components in exchange for other suitable components via the same interfaces. Furthermore, although the sheer scale and complexity of the MuMMI workflow and the capabilities it enables necessitates expert HPC usage, we have spent substantial effort in alleviating the barrier to entry for advanced and motivated users, focusing on the usual yet important challenges of deployment, portability, and extension.

At the highest level, the different components of MuMMI depend upon about 20 software and tools directly, whereas the size of the complete dependency tree exceeds 150 packages. Several dependencies are standard packages whereas others are small tools with little or no support. We address the challenges in porting and deploying the software stack using *Spack* [28] to streamline the process. Where necessary, we created additional packages and uploaded to the *Spack* repository [29] to have a consistent way to deploy MuMMI. We have successfully deployed MuMMI not only within large HPC environments but also on standard laptop computers (for testing and use of individual components).

With the goal of community adoption in mind, we are currently making progress toward an open-source release of MuMMI. Several independent components are already available open-source, *e.g.,* ddcMD [78], DynIm [14], Maestro [23], and Flux [3]. We make the MuMMI workflow portion available through GitHub[1] and *Spack*, which, in conjunction with the already open-sourced components, will allow expanding the applicability of this framework.

The MuMMI workflow is written in Python and is, therefore, portable and easy to incorporate in new projects, either piecewise or as a whole. Given an application scope (*e.g.,* tools to run simulations of different types), one can build upon the templates provided by the MuMMI workflow to customize for the specific use cases.

[1]https://github.com/mummi-framework

Table 1: MuMMI can seamlessly (re)start runs at different computational scales. This work utilized over 600,000 node hours on *Summit* using several runs at varying scales.

#nodes	wall-time	#runs	node hours
100	6 hours	5	3000
100	12 hours	3	3600
500	12 hours	3	18,000
1000	24 hours	20	480,000
4000	24 hours	1	96,000

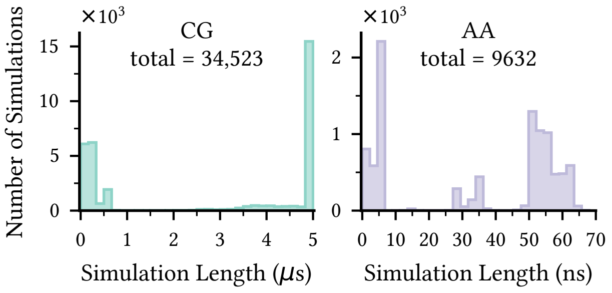

Figure 3: MuMMI enabled a large three-scale simulation of RAS-RAF-PM interactions probed using thousands of CG and AA simulations with varying lengths (as shown), and a single continuum simulation (not shown here) to cover large length- and time-scales that ran for over 20.5 ms.

For example, the abstract *Job Tracker* can be extended through inheritance and configuration files for individual job specifications (*e.g.,* commands and resources). Similarly, the abstract *Feedback Manager* can be extended to specify for the exact nature of feedback suited to the application (*e.g.,* how to read, interpret, and aggregate the data). The abstract data interfaces may be extended or additional interfaces may be created using the provided API for specific handling of nonstandard data types or additional types of database services.

5 SUCCESSES AND CHALLENGES AT SCALE

In order to demonstrate MuMMI's capabilities at large computational scale, we use *Summit* [54] — the second-most-powerful supercomputer in the world (and the most powerful in the US) at the time of this work [70]. *Summit* is the premier example of heterogeneous architecture at this scale: the machine has 4608 computational nodes, with each node comprising two IBM® POWER9™ CPUs with 22 cores each and six NVIDIA Volta™ V100 GPUs. Therefore, *Summit* is uniquely suited for the use and demonstration of MuMMI. The development and testing of MuMMI was done also on *Lassen* [47], a similar but smaller machine.

5.1 Summary of the Simulation Campaign

We ran a large multiscale simulation of RAS-RAF-PM interactions on *Summit* from Dec. 2020 through Mar. 2021. Over this period of about three months, we ran several jobs of lengths up to 24 hours to perform a single multiscale simulation campaign continued using checkpoint files and utilized a total of over 600,000 node hours.

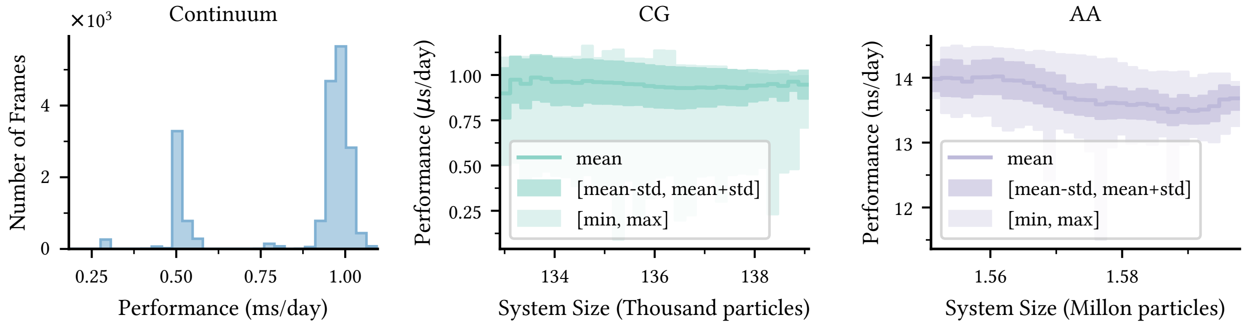

Figure 4: MuMMI delivered expected performance for the different scales of simulations. At continuum scale, the performance shows two prominent peaks that correspond to different number of CPU cores used. Performance of CG and AA simulations with respect to the system size show tight distributions around mean, although the slowest runs showed significant slow down.

Table 1 lists the different types and counts of jobs that were used for this run and also highlights the flexibility of the MuMMI workflow to seamlessly switch the scale up or down as needed, *e.g.,* restoring from a 500 node job to start a 1000 node one or vice versa.

As part of this execution, the continuum simulation produced 20,507 snapshots spaced 1 µs apart. The MuMMI workflow processed these snapshots and generated 6,828,831 patches, which were dynamically evaluated for novelty using ML. Based on novelty and availability of resources, 34,523 (= 0.5%) were selected and corresponding CG simulations were spawned. Through analysis of these CG simulations, 9,837,316 relevant CG frames were identified to be candidates for AA, of which, our sampling framework selected 9632 (= 0.098%) for spawning AA simulations. The CG and AA simulations were run to a maximum length of 5 µs and 50–65 ns, respectively (see Figure 3 for complete distributions).

In total, this simulation campaign accumulated over 20.5 ms of a single continuum simulation (1 µm × 1 µm large), 96.67 ms of CG trajectories (∽140 K particles), and 326 µs of AA trajectories (∽1.6 M atoms) — far ahead of any state-of-the-art simulation. For comparison, the earlier demonstration of MuMMI [24, 37] produced about 150 µs of continuum, 200 ms of CG, and no AA simulations. The other key difference is that our study conducted fewer but longer simulations; regardless, the MuMMI workflow can easily support different use cases.

Simulation Performance. Simulation performance was measured individually and compared against the expected peak performance (see Section 4.1). The observed performance using the three types of simulations through MuMMI is summarized in Figure 4.

In the case of the continuum simulation, the distribution of performance has three modes – each is associated with a certain size of the allocation (see Table 1). The typical configuration used 3600 CPU cores (150 nodes and 24 cores per node) and delivered the expected performance of ∽0.96 ms/day, whereas scaled-down performance was obtained using fewer CPU cores (100 and 500 node runs). The performance of CG and AA simulations depend upon the size of the corresponding system (the number of particles). In the case of CG, about one third into the simulation, we identified an issue with our software stack that ddcMD was compiled with an incompatible version of MPI, causing it to deliver almost 20% less than the benchmark [78]. Full performance was achieved after the fix; nevertheless, the overall distribution (shown in the figure) remains suboptimal. The performance of AA simulations (also highlighted in the figure) matches closely the AMBER benchmark measured outside of MuMMI.

5.2 Performance and Scaling of the Workflow

Given multiple types of simulations, our framework facilitates flexible configuration of the resource set, *e.g.,* a typical run used 60%–80% of the total GPUs for CG whereas the remaining were assigned to AA. To measure the performance of the workflow, we report the occupancy of resources in terms of the (percent of) time each GPU and CPU was working. MuMMI's profiling mechanism gathers the number of running and pending jobs every few minutes (for most of this campaign, profiling frequency was 10 min). Given the resource requirement for each job type, it is then straightforward to gather the number of occupied and unoccupied resources.

Figure 5 shows the distribution of the resource occupancy normalized with respect to the total size of the resource set (to account for the different sizes of allocations). The plot demonstrates an excellent GPU occupancy — 98% of all available GPUs were allocated for more than 83% of the total time (captured as profile events), with an average GPU occupancy of 93.73% and a median of 99.93%. The CPU occupancy, on the other hand, was lower, with most of the time, only about 50% allocated — on an average 54.12% allocated with a median occupancy of 50.48%. As before [24], this low average utilization is indicative of an application requirement, where CPU jobs are to be scheduled only when needed to prevent simulations of stale configurations (discussed in Task 3 in Section 4.4). Nevertheless, when needed (*i.e.,* when there are enough CG and AA configurations to be set up), the CPU occupancy reaches 100%.

Job Scheduling. By design, once the machine is loaded, MuMMI frequently (every few minutes) polls the system and replaces any finished/failed jobs immediately and, therefore, maintains high occupancy for long periods of time. One potential bottleneck is the initial time needed to fully load the machine. In principle, MuMMI

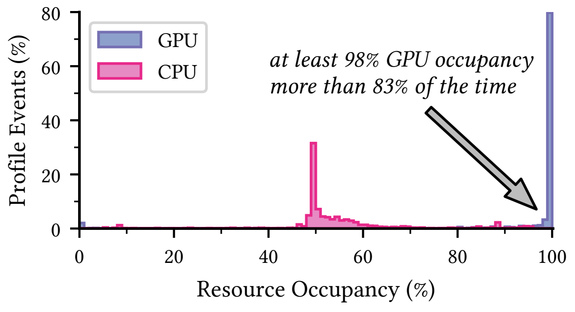

Figure 5: MuMMI facilitates excellent resource occupancy. Aggregating the profiles (computed every 10 mins) over all runs shows that the GPU occupancy was over 98% for more than 83% of the total time; CPU occupancy is low due to the need of the simulation.

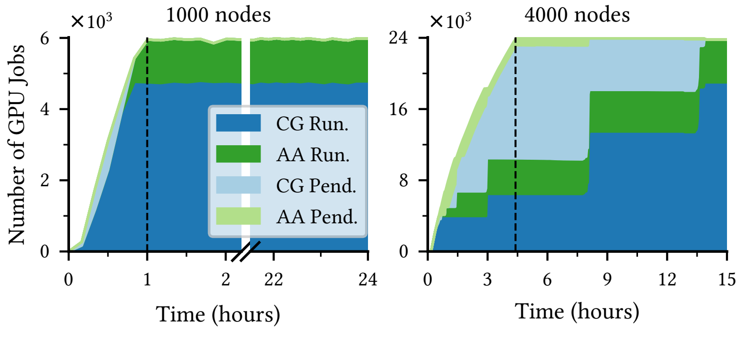

Figure 6: MuMMI supports high job submission rates. We configured our runs to submit ~100 jobs/min. Whereas a typical 1000-node run took only an hour to load, our scaling run (using 4000 nodes) revealed some scheduling bottlenecks where the submitted jobs took much longer to run. For simplicity, the figure shows GPU jobs only.

is capable of submitting several hundred jobs per minute through Maestro [23]. In practice, however, several other parameters affect the throughput, *e.g.,* the I/O throughput (submit scripts are written to disk), network bandwidth, OS limitations, and scheduler responsiveness. For most parts of this campaign, we specifically throttled the rate of submission to prevent overloading the job scheduler.

Figure 6 shows the history of job scheduling for a typical 1000-node run and highlights that the jobs are placed at a steady rate of about 100 jobs per min — an almost 3× improvement as compared to the previous work (2040 jobs in one hour [24]), not accounting for the fact that the jobs are now placed on specific GPUs rather than on complete nodes. Moving to scale (the 4000-node run), however, exposed some unanticipated bottlenecks in our scheduling framework. The figure shows that even at the same job submission rate, the scheduling in Flux happened in large chunks followed by large periods of inactivity, significantly diminishing overall throughput. Flux is currently one of the most performant resource and job manager available [3], and our scaling run, with its need to rapidly schedule 24,000 jobs across a resource set of hundreds of thousands of resources (including nodes, GPUs, CPU cores, sockets, and hardware threads) created an unprecedented stress test to evaluate the next generation of job managers.

Strategies for Further Scaling. We are currently working with the Flux team to devise strategies to scale the scheduling performance required for unbundled scheduling scheme. In the version of Flux used for this campaign, Flux's queue manager ($\mathcal{Q}$) and resource graph matcher ($\mathcal{R}$) communicate synchronously. Our scaling run exposed this bottleneck where $\mathcal{Q}$ spends the bulk of its time handling new job submissions as opposed to forwarding jobs to $\mathcal{R}$. We have since addressed this limitation by making this communication asynchronous. More crucially, we discovered that $\mathcal{R}$ essentially traverses the resource graph (modeling the resources managed by Flux) in its entirety for each job, particularly in the beginning when there are many vacant resources, creating "too many choices". We solved this problem by introducing a *first-match* policy that assigns the first matching resource set to a job greedily. Although an aggressive policy like this may not be suitable for batch job scheduling, it is well-suited for a workflow like MuMMI that contains significant high-throughput loads in its job mix.

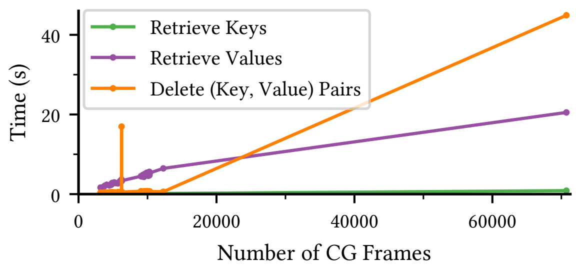

Figure 7: A Redis database (cluster of 20 nodes) facilitates fast feedback through real-time, in-memory data communication, as illustrated for the three types of queries made during CG-to-Continuum feedback.

Under Flux's emulated environment with a resource graph configuration similar to 4000 *Summit* nodes and the same job mix (24,000 jobs with 1 GPU and 3 CPU cores each, and 1 job with 150 nodes, each with 24 cores), we measured a 670× improvement in the performance. Nevertheless, an additional demonstration at full scale is not possible due to unavailability of computational resources and is, therefore, beyond the scope of this paper.

Feedback and I/O Performance. A key requirement for our workflow was to enable fast feedback loops — ideally, each iteration taking less than 10 min (the previous work provided an unsatisfactory frequency of two hours). By design, the two types of feedback have different computational challenges; therefore, we discuss different types of performance results for the two cases.

The CG-to-Continuum feedback is expected to process tens of thousands of CG frames in each iteration, where the processing itself is fast (vectorized additions of small Numpy arrays). Here, the performance is limited by the I/O throughput, *i.e.,* identifying the new data, loading the relevant data, and moving the processed data out of the relevant namespace. In this case, using an in-memory database facilitates fast I/O. We used MuMMI's *redis* interface for feedback during the scaling run (4000 nodes) and configured the database to use 20 nodes and mapped all compute nodes

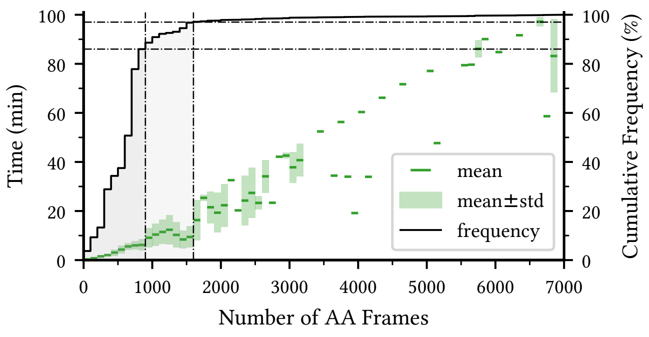

Figure 8: Processing of frames for AA-to-CG feedback requires calling external modules through system calls, posing computational overheads. Nevertheless, through several strategies, we successfully contain the processing time to within about 10 minutes for over 97% of the iterations.

randomly to these 20 Redis nodes. At this scale, MuMMI achieved a throughput of ∽10,000 queries (retrieval of keys) and deletions (of key-value pairs), and ∽2000 reads (retrieval of values) per second. Figure 7 details the performance of our *redis* interface and also highlights some aberrant behavior — likely due to network congestion, a key challenge for parallel HPC applications [13, 40]. Note that one data point shows an outlier behavior (70,000 frames) — a by-product of the intended early termination (for controlled shutdown) of the WM in the prior run, which causes unprocessed frames to accumulate.

In contrast, for the AA-to-CG feedback, I/O requirements are modest as it processes an order of magnitude fewer frames per iteration (there are fewer AA simulations, and AA frames are further filtered for eligibility for feedback). Each processing step, however, is expensive. Specifically, each AA frame is processed for ∽2 s through subprocess calls to an external program — each such call costs additional time (*e.g.,* due to the OS needing to spawn a new process and loading the required Python modules). Performing hundreds to thousands of such processing is challenging at scale. Therefore, at the workflow level, the feedback process was split into different phases for performance optimization, and suitable process pools and localized temporary files were used. Such improvements allowed bounding the processing time to within the target time limit. Figure 8 shows the distribution of time taken for each feedback iteration with respect to the number of AA frames processed, as well as the cumulative distribution of the number of frames. The figure shows that more than 97% of the feedback iterations finished within 10 minutes on average. In the few cases where more than 1600 frames had to be processed, we did not meet the target, but the performance scaled linearly. The figure also highlights high performance variability — a well-known concern in HPC [11, 49].

During this campaign, we used the *pytaridx* package to aggregate files of similar function into archives, *e.g.,* patches, snapshots, analysis, and RDFs. By the end, we had compiled *over 1 billion files* (1,034,232,900, to be precise) across 114,552 tar archives — a 9000× reduction in the number of files (and inodes) while retaining efficient random access. Taridx files also scale in size and number of constituent files — the largest in our simulation contains 6,723,600 files totaling about 455 GB. Reading from a tar file provides a throughput of ∽575 files/s or ∽87.56 MB/s (at ∽156 KB/file).

6 CONCLUSION AND DISCUSSION

MuMMI represents a generic design ideology of coupling two scales of interest using ML and *in situ* feedback [24, 37]. Here, we showcase how this powerful idea can be realized to encompass additional scales and models through pairwise coupling. We use a complex scientific inquiry — an exploration of the interactions of RAS-RAF protein complex with the PM in the context of cancer initiation mechanism – and present the *first-ever* demonstration of a massive ML-driven simulation campaign with *three resolution scales.*

We present technical innovations that extend the MuMMI workflow into a *generalizable* and *scalable* framework. Using *Summit*, currently the second-most-powerful machine in the world, we demonstrate the scaling of our infrastructure — several manifolds of improvement in all key components, including job scheduling, data management, and throughput of ML and feedback. In particular, achievements such as delivering an almost-perfect resource occupancy for most of the campaign and the management of over a billion files in total not only highlight the tremendous capacity of our workflow but also indicate how the boundaries of large multiscale ensembles will be pushed in the near future. We also discuss how to use our framework and, with forthcoming open-source release, hope for adoption of our technology by other applications. In particular, by following a two-part (coordination and application) model, we have paved way to easily substitute the domain-specific components in MuMMI, which has enabled us to utilize MuMMI for another application: namely, understanding biological interactions of neuroreceptors. Our efforts are geared towards improving the capabilities of large multiscale simulation campaigns to leverage modern, heterogeneous computing architectures, especially in anticipation of the upcoming Exascale machines [7, 48].

Outlook and Learnings at Scale

Whereas every technological breakthrough brings accomplishments in the form of new capabilities and insights, overcoming current obstacles usually exposes new types of limitations and often breaks assumptions of HPC paradigms and infrastructure. The process of running a three-month long scientific campaign posed several challenges worthy of a broader discussion.

Evolving Infrastructure Needs. MuMMI, and other similar autonomous workflows, rely upon the ability to dynamically co-schedule jobs. The full power of such workflows is only realized when dynamic co-scheduling is available natively; otherwise, workflows must still operate within static allocations requested through the system scheduler, tying up large portions of the machine for long periods of time. Furthermore, functioning within independent static allocations is wasteful since each allocation must suffer through slow startup, Given that dynamic workflows aim to amortize this cost over long periods, facilitating new infrastructure, in the form of either longer allocation limits and/or elastic resource availability, should be considered broadly as an emerging need.

Responsible Use of Shared Resources. Nevertheless, until such elastic resource availability becomes standard, MuMMI can still support effective and responsible use of partial machines with little overhead to other, nonworkflow jobs on the machine. Utilizing Flux allows us to mitigate the impact on the native system scheduler by encapsulating MuMMI's many-small-tasks paradigm as a large, singular job alike other conventional HPC jobs. This approach not only improves the workflow's overall job throughput, but also reduces impact on traditional HPC jobs. Shared filesystems are another resource that jobs usually contend for. MuMMI employs a conscious mix of the shared filesystem and local on-node RAM disk, which alleviates its footprint by reducing frequency of high-bandwidth file I/O operations that may interfere with other jobs.

Collaborative Ecosystems. Prior simulation art has primarily been based on the notion of large MPI-based ensembles executed by one or few expert(s). However, the shift to dynamic workflows with various dependent or independent components instead necessitates larger teams with a much more versatile skill set. The transition from single-user to multidisciplinary teams breaks the current HPC ecosystem that ties data and infrastructure access, job management, and other facilities to a single user. Although Unix groups provide mitigation for shared deployments and file access, the current computing ecosystem lacks solutions that make the process of managing such dynamic workflows seamless for multiple users. In particular, a modality of a group ecosystem needs to emerge that allows for groups of individuals to perform job management (monitoring, submission, cancellation, *etc.*). One attractive solution is the facilitation of "service accounts" by computing centers to allow a set of users uniform access over jobs while maintaining the required privacy and security measures.

Software Co-design. The increasing need for dynamic workflows further emphasizes the value of co-designing scalable frameworks that provide each other the drivers for innovation by pushing the limits in new ways. By drawing increasingly complex interconnections (*e.g.,* of tools, technologies, and resources), such workflows expose hidden conflicts, vulnerabilities, and/or performance bottlenecks. As an example, our collaboration with the Flux team provided immense value to both groups as our target provides a concrete problem at a scale previously not exposed to the scheduling mechanism, resulting in current and future scaling of performance. Synergistic feedback and collaboration between teams, especially across domains, is key to overcoming the ever-present "next hurdle".

The Next Leap. There is a growing need for developing *persistent workflows* to seamlessly connect software stacks and data services across allocations and even across clusters, possibly integrated with cloud and container technologies. Persistent workflows will allow leveraging supercomputers more effectively by decoupling compute from the system state and dynamism of the workflow. In future iterations of MuMMI, we envision a persistent workflow that can coordinate variable sized allocations as resources become available on different clusters. Exploring the shift to this paradigm is the next technological leap, crucial to facilitate the highest utility of HPC.

ACKNOWLEDGMENTS

This work has been supported in part by the Joint Design of Advanced Computing Solutions for Cancer (JDACS4C) program established by the U.S. Department of Energy (DOE) and the National Cancer Institute (NCI) of the National Institutes of Health (NIH). We thank the entire JDACS4C Pilot 2 team, particularly the Pilot 2 leads Fred Streitz and Dwight V. Nissley, for their support and helpful discussion.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, Los Alamos National Laboratory under Contract DE-AC5206NA25396, and Oak Ridge National Laboratory under contract DE-AC05-00OR22725. This research used resources of the Oak Ridge Leadership Computing Facility (OLCF), which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. For computing time, we thank the Advanced Scientific Computing Research Leadership Computing Challenge (ALCC) for time on *Summit* and Livermore Computing (LC) and Livermore Institutional Grand Challenge for time on *Lassen*. For computing support, we thank OLCF and LC staff. For data management support, we thank Bruce D'Amora, Lars Schneidenbach, Claudia Misale, and Carlos Costa.

LLNL Release number: LLNL-CONF-821231.

REFERENCES

[1] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. 2015. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* 1-2 (Sept. 2015), 19–25. https://doi.org/10.1016/j.softx.2015.06.001

[2] Brian M. Adams, Lara E. Bauman, William J. Bohnhoff, Keith R. Dalbey, Mohamed S. Ebeida, John P. Eddy, Michael S. Eldred, Patricia D. Hough, Kenneth T. Hu, John D. Jakeman, J. Adam Stephens, Laura P. Swiler, Dena M. Vigil, and Timothy M. Wildey. 2009. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User's Manual.* Sandia National Laboratory.

[3] Dong H. Ahn, Ned Bass, Albert Chu, Jim Garlick, Mark Grondona, Stephen Herbein, Helgi I. Ingólfsson, Joseph Koning, Tapasya Patki, Thomas R.W. Scogland, Becky Springmeyer, and Michela Taufer. 2020. Flux: Overcoming scheduling challenges for exascale workflows. *Future Generation Computer Systems* 110 (2020), 202–213. https://doi.org/10.1016/j.future.2020.04.006

[4] Riccardo Alessandri, Paulo C. T. Souza, Sebastian Thallmair, Manuel N. Melo, Alex H. de Vries, and Siewert J. Marrink. 2019. Pitfalls of the Martini Model. *Journal of Chemical Theory and Computation* 15, 10 (2019), 5448–5460. https://doi.org/10.1021/acs.jctc.9b00473 PMID: 31498621.

[5] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. 2004. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management, 2004.* IEEE, 423–424. https://doi.org/10.1109/SSDM.2004.1311241

[6] Nojood A. Altwaijry, Michael Baron, David W. Wright, Peter V. Coveney, and Andrea Townsend-Nicholson. 2017. An Ensemble-Based Protocol for the Computational Prediction of Helix–Helix Interactions in G Protein-Coupled Receptors using Coarse-Grained Molecular Dynamics. *Journal of Chemical Theory*

and Computation 13, 5 (2017), 2254–2270. https://doi.org/10.1021/acs.jctc.6b01246

[7] Argonne National Laboratory. 2021. *Aurora*. Retrieved March, 2021 from https://www.alcf.anl.gov/aurora

[8] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S. Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin Wozniak, Ian Foster, Mike Wilde, and Kyle Chard. 2019. Parsl: Pervasive Parallel Programming in Python. In *28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. https://doi.org/10.1145/3307681.3325400

[9] Tal Ben-Nun, Todd Gamblin, D. S. Hollman, Hari Krishnan, and Chris J. Newburn. 2020. Workflows are the New Applications: Challenges in Performance, Portability, and Productivity. In *IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. 57–69. https://doi.org/10.1109/P3HPC51967.2020.00011

[10] Robert B. Best, Xiao Zhu, Jihyun Shim, Pedro E. M. Lopes, Jeetain Mittal, Michael Feig, and Alexander D. MacKerell. 2012. Optimization of the Additive CHARMM All-Atom Protein Force Field Targeting Improved Sampling of the Backbone Φ, Ψ and Side-Chain $\chi1$ and $\chi2$ Dihedral Angles. *Journal of Chemical Theory and Computation* 8, 9 (2012), 3257–3273. https://doi.org/10.1021/ct300400x PMID: 23341755.

[11] Abhinav Bhatele, Jayaraman J. Thiagarajan, Taylor Groves, Rushil Anirudh, Staci A. Smith, Brandon Cook, and David K. Lowenthal. 2020. The Case of Performance Variability on Dragonfly-based Systems. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 896–905. https://doi.org/10.1109/IPDPS47924.2020.00096

[12] Harsh Bhatia, Timothy S. Carpenter, Helgi I. Ingólfsson, Gautham Dharuman, Piyush Karande, Shusen Liu, Tomas Oppelstrup, Chris Neale, Felice C. Lightstone, Brian Van Essen, James N. Glosli, and Peer-Timo Bremer. 2021. Machine Learning Based Dynamic-Importance Sampling for Adaptive Multiscale Simulations. *Nature Machine Intelligence* 3 (2021), 401–409. https://doi.org/10.1038/s42256-021-00327-w

[13] Harsh Bhatia, Nikhil Jain, Abhinav Bhatele, Yarden Livnat, Jens Domke, Valerio Pascucci, and Peer-Timo Bremer. 2018. Interactive Investigation of Traffic Congestion on Fat-Tree Networks Using TreeScope. *Computer Graphics Forum* 37, 3 (2018), 561–572. https://doi.org/10.1111/cgf.13442

[14] Harsh Bhatia and Joseph Y. Moon. 2020. Dynamic-Importance Sampling. https://github.com/LLNL/dynim.

[15] J. Borgdorff, M. Ben Belgacem, C. Bona-Casas, L. Fazendeiro, D. Groen, O. Hoenen, A. Mizeranschi, J.L. Suter, D. Coster, P.V. Coveney, W. Dubitzky, A.G. Hoekstra, P. Strand, and B. Chopard. 2014. Performance of distributed multiscale simulations. *Phil. Trans. R. Soc. A* 372 (2014), 20130407. https://doi.org/10.1098/rsta.2013.0407

[16] Hans-Joachim Bungartz, Florian Lindner, Bernhard Gatzhammer, Miriam Mehl, Klaudius Scheufele, Alexander Shukaev, and Benjamin Uekermann. 2016. preCICE – A fully parallel library for multi-physics surface coupling. *Computers & Fluids* 141 (2016), 250–258. https://doi.org/10.1016/j.compfluid.2016.04.003

[17] Lorenzo Casalino, Abigail Dommer, Zied Gaieb, Emilia P. Barros, Terra Sztain, Surl-Hee Ahn, Anda Trifan, Alexander Brace, Anthony Bogetti, Heng Ma, Hyungro Lee, Matteo Turilli, Syma Khalid, Lillian Chong, Carlos Simmerling, David J. Hardy, Julio D. C. Maia, James C. Phillips, Thorsten Kurth, Abraham Stern, Lei Huang, John McCalpin, Mahidhar Tatineni, Tom Gibbs, John E. Stone, Shantenu Jha, Arvind Ramanathan, and Rommie E. Amaro. 2020. AI-Driven Multiscale Simulations Illuminate Mechanisms of SARS-CoV-2 Spike Dynamics. *bioRxiv* (2020). https://doi.org/10.1101/2020.11.19.390187

[18] David A. Case, Thomas E. Cheatham III, Tom Darden, Holger Gohlke, Ray Luo, Kenneth M. Merz Jr., Alexey Onufriev, Carlos Simmerling, Bing Wang, and Robert J. Woods. 2005. The Amber biomolecular simulation programs. *Journal of Computational Chemistry* 26, 16 (2005), 1668–1688. https://doi.org/10.1002/jcc.20290

[19] Bastien Chopard, Joris Borgdorff, and Alfons Hoekstra. 2014. A framework for multi-scale modelling. *Philosophical Transactions of The Royal Society A* 372 (2014), 20130378. https://doi.org/10.1098/rsta.2013.0378

[20] Anthony Craig, Sophie Valcke, and Laure Coquart. 2017. Development and performance of a new version of the OASIS coupler, OASIS3-MCT_3.0. *Geoscientific Model Development* 10, 9 (2017), 3297–3308. https://doi.org/10.5194/gmd-10-3297-2017

[21] Tamara L. Dahlgren, David Domyancic, Scott Brandon, Todd Gamblin, John Gyllenhaal, Rao Nimmakayala, and Richard Klein. 2015. Poster: Scaling uncertainty quantification studies to millions of jobs. In *Proceedings of the 27th ACM/IEEE International Conference for High Performance Computing and Communications Conference (SC)*.

[22] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, and Kent Wenger. 2015. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems* 46 (2015), 17–35. https://doi.org/10.1016/j.future.2014.10.008

[23] Francesco Di Natale. 2017. Maestro Workflow Conductor. https://github.com/LLNL/maestrowf.

[24] Francesco Di Natale, Harsh Bhatia, Timothy S. Carpenter, Chris Neale, Sara Kokkila Schumacher, Tomas Oppelstrup, Liam Stanton, Xiaohua Zhang, Shiv Sundram, Thomas R. W. Scogland, Gautham Dharuman, Michael P. Surh, Yue Yang, Claudia Misale, Lars Schneidenbach, Carlos Costa, Changhoan Kim, Bruce D'Amora, Sandrasegaram Gnanakaran, Dwight V. Nissley, Fred Streitz, Felice C. Lightstone, Peer-Timo Bremer, James N. Glosli, and Helgi I. Ingólfsson. 2019. A Massively Parallel Infrastructure for Adaptive Multiscale Simulations: Modeling RAS Initiation Pathway for Cancer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '19)*. ACM, New York, NY, USA, Article 57, 16 pages. https://doi.org/10.1145/3295500.3356197

[25] Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, Franck Cappello, Barbara Chapman, Xuebin Chi, Alok Choudhary, Sudip Dosanjh, Thom Dunning, Sandro Fiore, Al Geist, Bill Gropp, Robert Harrison, Mark Hereld, Michael Heroux, Adolfy Hoisie, Koh Hotta, Zhong Jin, Yutaka Ishikawa, Fred Johnson, Sanjay Kale, Richard Kenway, David Keyes, Bill Kramer, Jesus Labarta, Alain Lichnewsky, Thomas Lippert, Bob Lucas, Barney Maccabe, Satoshi Matsuoka, Paul Messina, Peter Michielse, Bernd Mohr, Matthias S. Mueller, Wolfgang E. Nagel, Hiroshi Nakashima, Michael E Papka, Dan Reed, Mitsuhisa Sato, Ed Seidel, John Shalf, David Skinner, Marc Snir, Thomas Sterling, Rick Stevens, Fred Streitz, Bob Sugar, Shinji Sumimoto, William Tang, John Taylor, Rajeev Thakur, Anne Trefethen, Mateo Valero, Aad van der Steen, Jeffrey Vetter, Peg Williams, Robert Wisniewski, and Kathy Yelick. 2011. The International Exascale Software Project roadmap. *The International Journal of High Performance Computing Applications* 25, 1 (2011), 3–60. https://doi.org/10.1177/1094342010391989

[26] Florent Duchaine, Stéphan Jauré, Damien Poitou, Eric Quémerais, Gabriel Staffelbach, Thierry Morel, and Laurent Gicquel. 2015. Analysis of high performance conjugate heat transfer with the OpenPALM coupler. *Computational Science & Discovery* 8, 1 (July 2015), 015003. https://doi.org/10.1088/1749-4699/8/1/015003

[27] Ernest J. Friedman-Hill, Edward L. Hoffman, Marcus J. Gibson, Robert L. Clay, and Kevin H. Olson. 2015. *Incorporating Workflow for V&V/UQ in the Sandia Analysis Workbench.* Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[28] Todd Gamblin, Matthew LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral. 2015. The Spack Package Manager: Bringing Order to HPC Software Chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Austin, Texas) *(SC '15)*. ACM, New York, NY, USA, Article 40, 12 pages. https://doi.org/10.1145/2807591.2807623

[29] Todd Gamblin and The Spack Team. 2020. Spack. https://github.com/spack/spack.

[30] James N. Glosli, David F. Richards, Kyle J. Caspersen, Robert E. Rudd, John A. Gunnels, and Frederick H. Streitz. 2007. Extending Stability Beyond CPU Millennium: A Micron-scale Atomistic Simulation of Kelvin-Helmholtz Instability. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing* (Reno, Nevada). ACM, New York, NY, USA, Article 58, 11 pages. https://doi.org/10.1145/1362622.1362700

[31] John M.A. Grime, James F. Dama, Barbie K. Ganser-Pornillos, Cora L. Woodward, Grant J. Jensen, Mark Yeager, and Gregory A. Voth. 2016. Coarse-grained simulation reveals key features of HIV-1 capsid self-assembly. *Nature Communications* 7 (2016), 11568. https://doi.org/10.1038/ncomms11568

[32] Alfons Hoekstra, Bastien Chopard, and Peter Coveney. 2014. Multiscale modelling and simulation: A position paper. *Philosophical Transactions of The Royal Society A* 372 (2014), 20130377. https://doi.org/10.1098/rsta.2013.0377

[33] Tsuyoshi Ichimura, Kohei Fujita, Takuma Yamaguchi, Akira Naruse, Jack C. Wells, Thomas C. Schulthess, Tjerk P. Straatsma, Christopher J. Zimmer, Maxime Martinasso, Kengo Nakajima, Muneo Hori, and Lalith Maddegedara. 2018. A Fast Scalable Implicit Solver for Nonlinear Time-evolution Earthquake City Problem on Low-ordered Unstructured Finite Elements with Artificial Intelligence and Transprecision Computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. IEEE Press, Piscataway, NJ, USA, 627–637. https://doi.org/10.1109/SC.2018.00052

[34] Helgi I. Ingólfsson, Harsh Bhatia, Talia Zeppelin, W. F. Drew Bennett, Kristy A. Carpenter, Pin-Chia Hsu, Gautham Dharuman, Peer-Timo Bremer, Birgit Schiøtt, Felice C. Lightstone, and Timothy S. Carpenter. 2020. Capturing Biologically Complex Tissue-Specific Membranes at Different Levels of Compositional Complexity. *The Journal of Physical Chemistry B* 124, 36 (2020), 7819–7829. https://doi.org/10.1021/acs.jpcb.0c03368 PMID: 32790367.

[35] Helgi I. Ingólfsson, Timothy S. Carpenter, Harsh Bhatia, Peer-Timo Bremer, Siewert J. Marrink, and Felice C. Lightstone. 2017. Computational Lipidomics of the Neuronal Plasma Membrane. *Biophysical Journal* 113, 10 (Nov. 2017), 2271–2280. https://doi.org/10.1016/j.bpj.2017.10.017

[36] Helgi I. Ingólfsson, Cesar A. Lopez, Jaakko J. Uusitalo, Djurre H. de Jong, Srinivasa M. Gopal, Xavier Periole, and Siewert J. Marrink. 2014. The power of coarse graining in biomolecular simulations. *WIREs Computational Molecular Science* 4, 3 (2014), 225–248. https://doi.org/10.1002/wcms.1169

[37] Helgi I. Ingólfsson, Chris Neale, Timothy S. Carpenter, Rebika Shrestha, Cesar A López, Timothy H. Tran, Tomas Oppelstrup, Harsh Bhatia, Liam G. Stanton, Xiaohua Zhang, Shiv Sundram, Francesco Di Natale, Animesh Agarwal, Gautham

Dharuman, Sara I. L. Kokkila Schumacher, Thomas Turbyville, Gulcin Gulten, Que N. Van, Debanjan Goswami, Frantz Jean-Francios, Constance Agamasu, De Chen, Jeevapani J. Hettige, Timothy Travers, Sumantra Sarkar, Michael P. Surh, Yue Yang, Adam Moody, Shusen Liu, Brian C. Van Essen, Arthur F. Voter, Arvind Ramanathan, Nicolas W. Hengartner, Dhirendra K. Simanshu, Andrew G. Stephen, Peer-Timo Bremer, S. Gnanakaran, James N. Glosli, Felice C. Lightstone, Frank McCormick, Dwight V. Nissley, and Frederick H. Streitz. 2020. Machine Learning-driven Multiscale Modeling Reveals Lipid-Dependent Dynamics of RAS Signaling Proteins. (2020). https://doi.org/10.21203/rs.3.rs-50842/v1 Preprint.

[38] Sam Ade Jacobs, Tim Moon, Kevin McLoughlin, Derek Jones, David Hysom, Dong H. Ahn, John Gyllenhaal, Pythagoras Watson, Felice C. Lightstone, Jonathan E. Allen, Ian Karlin, and Brian Van Essen. 2020. Enabling Rapid COVID-19 Small Molecule Drug Design Through Scalable Deep Learning of Generative Models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20)*. ACM, New York, NY, USA. https://doi.org/10.1177/10943420211010930 Finalist for the 2020 Gordon Bell Special Prize.

[39] Anubhav Jain, Shyue Ping Ong, Wei Chen, Bharat Medasani, Xiaohui Qu, Michael Kocher, Miriam Brafman, Guido Petretto, Gian-Marco Rignanese, Geoffroy Hautier, Daniel Gunter, and Kristin A. Persson. 2015. FireWorks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience* 27, 17 (2015), 5037–5059. https://doi.org/10.1002/cpe.3505 CPE-14-0307.R2.

[40] Nikhil Jain, Abhinav Bhatele, Sam White, Todd Gamblin, and Laxmikant V. Kale. 2016. Evaluating HPC Networks via Simulation of Parallel Workloads. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 154–165. https://doi.org/10.1109/SC.2016.13

[41] Hervé Jégou, Matthijs Douze, Jeff Johnson, and Lucas Hosseini. [n.d.]. FAISS. https://github.com/facebookresearch/faiss.

[42] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* (2019). https://doi.org/10.1109/TBDATA.2019.2921572

[43] Dirk Kessler, Michael Gmachl, Andreas Mantoulidis, Laetitia J. Martin, Andreas Zoephel, Moriz Mayer, Andreas Gollner, David Covini, Silke Fischer, Thomas Gerstberger, Teresa Gmaschitz, Craig Goodwin, Peter Greb, Daniela Häring, Wolfgang Hela, Johann Hoffmann, Jale Karolyi-Oezguer, Petr Knesl, Stefan Kornigg, Manfred Koegl, Roland Kousek, Lyne Lamarre, Franziska Moser, Silvia Munico-Martinez, Christoph Peinsipp, Jason Phan, Jörg Rinnenthal, Jiqing Sai, Christian Salamon, Yvonne Scherbantin, Katharina Schipany, Renate Schnitzer, Andreas Schrenk, Bernadette Sharps, Gabriella Siszler, Qi Sun, Alex Waterson, Bernhard Wolkerstorfer, Markus Zeeb, Mark Pearson, Stephen W. Fesik, and Darryl B. McConnell. 2019. Drugging an undruggable pocket on KRAS. *Proceedings of the National Academy of Sciences* 116, 32 (2019), 15823–15829. https://doi.org/10.1073/pnas.1904529116

[44] Kai J. Kohlhoff, Diwakar Shukla, Morgan Lawrenz, Gregory R. Bowman, David E. Konerding, Dan Belov, Russ B. Altman, and Vijay S. Pande. 2014. Cloud-based simulations on Google Exacycle reveal ligand modulation of GPCR activation pathways. *Nature Chemistry* 6, 1 (2014), 15–21. https://doi.org/10.1038/nchem.1821

[45] V.V. Krzhizhanovskaya, D. Groen, B. Bozak, and A.G. Hoekstra. 2015. Multiscale Modelling and Simulation Workshop: 12 Years of Inspiration. *Procedia Computer Science* 51 (2015), 1082–1087. https://doi.org/10.1016/j.procs.2015.05.268 International Conference On Computational Science, ICCS 2015.

[46] Redis Labs. 2018. Redis. https://redis.io.

[47] Lawrence Livermore National Laboratory. 2019. *Lassen*. Retrieved March, 2021 from https://hpc.llnl.gov/hardware/platforms/lassen

[48] Lawrence Livermore National Laboratory. 2021. *El Capitan*. https://www.llnl.gov/news/llnl-and-hpe-partner-amd-el-capitan-projected-worlds-fastest-supercomputer

[49] Boyang Li, Sudheer Chunduri, Kevin Harms, Yuping Fan, and Zhiling Lan. 2019. The Effect of System Utilization on Application Performance Variability. In *Proceedings of the 9th International Workshop on Runtime and Operating Systems for Supercomputers* (Phoenix, AZ, USA) *(ROSS '19)*. Association for Computing Machinery, New York, NY, USA, 11–18. https://doi.org/10.1145/3322789.3328743

[50] Umberto M.B. Marconi and Pedro Tarazona. 1999. Dynamic density functional theory of fluids. *The Journal of chemical physics* 110, 16 (1999), 8032–8044. https://doi.org/10.1063/1.478705

[51] Siewert J. Marrink, H. Jelger Risselada, Serge Yefimov, D. Peter Tieleman, and Alex H. de Vries. 2007. The MARTINI Force Field: Coarse Grained Model for Biomolecular Simulations. *The Journal of Physical Chemistry B* 111, 27 (July 2007), 7812–7824. https://doi.org/10.1021/jp071097f

[52] Manuel N. Melo, Clément Arnarez, Hendrik Sikkema, Neeraj Kumar, Martin Walko, Herman J. C. Berendsen, Armagan Kocer, Siewert J. Marrink, and Helgi I. Ingólfsson. 2017. High-Throughput Simulations Reveal Membrane-Mediated Effects of Alcohols on MscL Gating. *Journal of the American Chemical Society* 139, 7 (Feb. 2017), 2664–2671. https://doi.org/10.1021/jacs.6b11091

[53] Misako Nagasaka, Yiwei Li, Ammar Sukari, Sai-Hong Ignatius Ou, Mohammed Najeeb Al-Hallak, and Asfar S. Azmi. 2020. KRAS G12C Game of Thrones, which direct KRAS inhibitor will claim the iron throne? *Cancer Treatment Reviews* 84 (2020), 101974. https://doi.org/10.1016/j.ctrv.2020.101974

[54] Oak Ridge National Laboratory. 2019. *Summit*. Retrieved March, 2021 from https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit

[55] Alexander J. Pak, John M. A. Grime, Prabuddha Sengupta, Antony K. Chen, Aleksander E.P. Durumeric, Anand Srivastava, Mark Yeager, John A.G. Briggs, Jennifer Lippincott-Schwartz, and Gregory A. Voth. 2017. Immature HIV-1 lattice assembly dynamics are regulated by scaffolding from nucleic acid and the plasma membrane. *Proceedings of the National Academy of Sciences* 114, 47 (2017), E10056–E10065. https://doi.org/10.1073/pnas.1706600114

[56] Albert C. Pan, Daniel Jacobson, Konstantin Yatsenko, Duluxan Sritharan, Thomas M. Weinreich, and David E. Shaw. 2019. Atomic-level characterization of protein–protein association. *Proceedings of the National Academy of Sciences* 116, 10 (2019), 4244–4249. https://doi.org/10.1073/pnas.1815431116

[57] J. Luc Peterson, Ben Bay, Joe Koning, Peter Robinson, Jessica Semler, Jeremy White, Rushil Anirudh, Kevin Athey, Peer-Timo Bremer, Francesco Di Natale, David Fox, Jim A. Gaffney, Sam A. Jacobs, Bogdan Kustowski Bhavya Kailkhura, Steven Langer, Brian Spears, Jayaraman J. Thiagarajan, Brian Van Essen, and Jae-Seung Yeom. 2019. Merlin: Enabling Machine Learning-Ready HPC Ensembles. https://arxiv.org/abs/1912.02892

[58] Jayson L. Peterson, Kelli D. Humbird, John E. Field, Scott T. Brandon, Steve H. Langer, Ryan C. Nora, Brian K. Spears, and Paul T. Springer. 2017. Zonal Flow Generation in Inertial Confinement Fusion Implosions. *Physics of Plasmas* 24, 3 (2017), 032702. https://doi.org/10.1063/1.4977912

[59] Ian A. Prior, Fiona E. Hood, and James L. Hartley. 2020. The Frequency of Ras Mutations in Cancer. *Cancer Research* 80, 14 (2020), 2969–2974. https://doi.org/10.1158/0008-5472.CAN-19-3682

[60] Benedict J. Reynwar, Gregoria Illya, Vagelis A. Harmandaris, Martin M. Müller, Kurt Kremer, and Markus Deserno. 2007. Aggregation and vesiculation of membrane proteins by curvature-mediated interactions. *Nature* 447, 7143 (2007), 461. https://doi.org/10.1038/nature05840

[61] Rob Farber. 2020. *Workflow Technologies impact SC20 Gorden Bell COVID-19 Award Winner and Two of the Three Finalists*. Retrieved May, 2021 from https://www.exascaleproject.org/workflow-technologies-impact-sc20-gordon-bell-covid-19-award-winner-and-two-of-the-three-finalists/

[62] Romelia Salomon-Ferrer, Andreas W. Götz, Duncan Poole, Scott Le Grand, and Ross C. Walker. 2013. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald. *Journal of Chemical Theory and Computation* 9, 9 (2013), 3878–3888. https://doi.org/10.1021/ct400314y PMID: 26592383.

[63] Marissa G. Saunders and Gregory A. Voth. 2013. Coarse-Graining Methods for Computational Biology. *Annual Review of Biophysics* 42, 1 (2013), 73–93. https://doi.org/10.1146/annurev-biophys-083012-130348

[64] David E. Shaw, Ron O. Dror, John K. Salmon, J.P. Grossman, Kenneth M. Mackenzie, Joseph A. Bank, Cliff Young, Martin M. Deneroff, Brannon Batson, Kevin J. Bowers, Edmond Chow, Michael P. Eastwood, Douglas J. Ierardi, John L. Klepeis, Jeffrey S. Kuskin, Richard H. Larson, Kresten Lindorff-Larsen, Paul Maragakis, Mark A. Moraes, Stefano Piana, Yibing Shan, and Brian Towles. 2009. Millisecond-scale Molecular Dynamics Simulations on Anton. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (Portland, Oregon) *(SC '09)*. New York, NY, USA, 1–11. https://doi.org/10.1145/1654059.1654126

[65] Michael R. Shirts, Christoph Klein, Jason M. Swails, Jian Yin, Michael K. Gilson, David L. Mobley, David A. Case, and Ellen D. Zhong. 2017. Lessons learned from comparing molecular dynamics engines on the SAMPL5 dataset. *Journal of Computer-Aided Molecular Design* 31, 1 (2017), 147–161. https://doi.org/10.1007/s10822-016-9977-1

[66] Dhirendra K. Simanshu, Dwight V. Nissley, and Frank McCormick. 2017. RAS Proteins and Their Regulators in Human Disease. *Cell* 170, 1 (June 2017), 17–33. https://doi.org/10.1016/j.cell.2017.06.009

[67] James Smith. 2017. *IBM Spectrum LSF*. IBM Corporation. https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_welcome/lsf_welcome.html

[68] Frederick H Streitz, James N Glosli, and Mehul V Patel. 2006. Beyond finite-size scaling in solidification simulations. *Physical Review Letters* 96, 22 (2006), 225701. https://doi.org/10.1103/PhysRevLett.96.225701

[69] Frederick H. Streitz, James N. Glosli, Mehul V. Patel, Bor Chan, Robert K. Yates, Bronis R. de Supinski, James Sexton, and John A. Gunnels. 2005. 100+ TFlop Solidification Simulations on BlueGene/L. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing* (Seattle, Washington) *(SC '05)*. ACM, New York, NY, USA.

[70] TOP500. 2020. *TOP500 Supercomputer Sites | November 2020*. Retrieved March, 2021 from https://www.top500.org/lists/top500/2020/11/

[71] Timothy Travers, Cesar A. López, Que N. Van, Chris Neale, Marco Tonelli, Andrew G. Stephen, and Sandrasegaram Gnanakaran. 2018. Molecular recognition of RAS/RAF complex at the membrane: Role of RAF cysteine-rich domain. *Scientific Reports* 8, 1 (May 2018), 8461. https://doi.org/10.1038/s41598-018-26832-4

[72] Vincent A. Voelz, Marcus Jäger, Shuhuai Yao, Yujie Chen, Li Zhu, Steven A. Waldauer, Gregory R. Bowman, Mark Friedrichs, Olgica Bakajin, Lisa J. Lapidus, Shimon Weiss, and Vijay S. Pande. 2012. Slow Unfolded-State Structuring in Acyl-CoA Binding Protein Folding Revealed by Simulation and Experiment. *Journal of the American Chemical Society* 134, 30 (2012), 12565–12577. https://doi.org/10.1021/ja302528z

[73] Gregory A. Voth. 2017. A Multiscale Description of Biomolecular Active Matter: The Chemistry Underlying Many Life Processes. *Accounts of Chemical Research* 50, 3 (March 2017), 594–598. https://doi.org/10.1021/acs.accounts.6b00572

[74] Tsjerk A. Wassenaar, Helgi I. Ingólfsson, Rainer A. Böckmann, D. Peter Tieleman, and Siewert J. Marrink. 2015. Computational Lipidomics with *insane* : A Versatile Tool for Generating Custom Membranes for Molecular Simulations. *Journal of Chemical Theory and Computation* 11, 5 (May 2015), 2144–2155. https://doi.org/10.1021/acs.jctc.5b00209

[75] Tsjerk A. Wassenaar, Kristyna Pluhackova, Rainer A. Böckmann, Siewert J. Marrink, and D. Peter Tieleman. 2014. Going Backward: A Flexible Geometric Approach to Reverse Transformation from Coarse Grained to Atomistic Models. *Journal of Chemical Theory and Computation* 10, 2 (2014), 676–690. https://doi.org/10.1021/ct400617g PMID: 26580045.

[76] Tsjerk A. Wassenaar, Kristyna Pluhackova, Anastassiia Moussatova, Durba Sengupta, Siewert J. Marrink, D. Peter Tieleman, and Rainer A. Böckmann. 2015. High-Throughput Simulations of Dimer and Trimer Assembly of Membrane Proteins. The DAFT Approach. *Journal of Chemical Theory and Computation* 11, 5 (May 2015), 2278–2291. https://doi.org/10.1021/ct5010092

[77] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2002. SLURM: Simple Linux Utility for Resource Management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 44–60. https://doi.org/10.1007/10968987_3

[78] Xiaohua Zhang, Shiv Sundram, Tomas Oppelstrup, Sara I. L. Kokkila-Schumacher, Timothy S. Carpenter, Helgi I. Ingólfsson, Frederick H. Streitz, Felice C. Lightstone, and James N. Glosli. 2020. ddcMD: A fully GPU-accelerated molecular dynamics program for the Martini force field. *The Journal of Chemical Physics* 153, 4 (2020), 045103. https://doi.org/10.1063/5.0014500

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We tested ddcMD (version 2020.1) molecular dynamics code on the Summit supercomputer (Oak Ridge National Lab) using SpectrumMPI v10.3.0.1-20190611-flux and NVIDIA CUDA library version 10.1.105. were used. Performance was tested using a representative system and subsequently verified during the simulation campaign (see Figure 4, graph labeled “CG”).

To test AMBER 2018, we computed the simulation time based on the output of all-atomistic simulations during our simulation campaigns, and the value representative of runs using the setup in the “Environmental Setup” section of these appendices. Output rate was determined by post-processing the timestamp of simulation output and computing a framerate. The results are presented in Figure 4 in the graph labeled “AA”.

Testing GridSim2D was done on Summit prior to the simulation campaign using the environment described in the “Experimental Setup” on a representative system. We independently verified our expected performance during our campaign on Summit and present the performance seen during our demonstrated campaign (see Figure 4, graph labeled “Continuum”).

Author-Created or Modified Artifacts:

```
Persistent ID: https://github.com/LLNL/maestrowf
Artifact name: Maestro Workflow Conductor
Citation of artifact: Di Natale, Francesco. Maestro
 ↪  Workflow Conductor. Computer software. Vers. 00.
 ↪  USDOE National Nuclear Security Administration
 ↪  (NNSA). 1 Jun. 2017. Web.

Persistent ID: https://github.com/LLNL/dynim
Artifact name: DynIm
Citation of artifact: Bhatia, Carpenter, Inglfsson,
 ↪  Dharuman, Karande, Liu, Oppelstrup, Neale,
 ↪  Lightstone, Van Essen, Glosli, and Bremer. 2021.
 ↪  Machine Learning Based Dynamic-Importance
 ↪  Sampling for Adaptive Multiscale Simulations.
 ↪  Nature Machine Intelligence (2021). In Press.

Persistent ID:
 ↪  https://github.com/XiaohuaZhangLLNL/mdanalysis
Artifact name: MDAnalysis (LLNL modified)
Citation of artifact: N/A

Persistent ID: https://github.com/LLNL/ddcMD
Artifact name: ddcMD
Citation of artifact: Zhang, Sundram, Oppelstrup,
 ↪  Kokkila-Schumacher, Carpenter, Inglfsson,
 ↪  Streitz, Lightstone, and Glosli. 2020. ddcMD: A
 ↪  fully GPU-accelerated molecular dynamics program
 ↪  for the Martini force field. The Journal of
 ↪  Chemical Physics 153, 4 (2020), 045103.
 ↪  https://doi.org/10.1063/5.001450014
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Summit, NVIDIA Volta v100, IBM POWER9 CPUs, GPFS

Operating systems and versions: Red Hat Enterprise Linux (RHEL) version 7.4

Compilers and versions: Python 3.7.7, gcc v7.4.0

Applications and versions: AMBER 2018, GROMACS 2019.06, MDAnalysis v0.16.2, ddcMD v2019.1, maestrowf v1.1.9dev0, DSSP v3.1.4, parmed v3.2.0, Redis v5.0.3, GridSim2D v2020-12-22-c3-final, pytaridx, flux-core v0.20.0, flux-sched v0.9.0, Martinize 2.6.3, insane (doi:10.1021/acs.jctc.5b00209)

Libraries and versions: boost v1.73.0, Theano v1.0.2, faiss v1.6.3, SpectrumMPI v10.3.0.1-20190611-flux, CUDA v10.1.105

Key algorithms: N/A

Input datasets and versions: N/A