Asynchronous Multi-Level Checkpointing: An Enabler of Reproducibility using Checkpoint History Analytics

Kevin Assogba
Rochester Institute of Technology
Rochester, NY, USA
kta7930@cs.rit.edu

Bogdan Nicolae
Argonne National Laboratory
Lemont, IL, USA
bnicolae@anl.gov

Hubertus Van Dam
Brookhaven National Laboratory
Upton, NY, USA
hvandam@bnl.gov

M. Mustafa Rafique
Rochester Institute of Technology
Rochester, NY, USA
mrafique@cs.rit.edu

ABSTRACT

High-performance computing applications are increasingly integrating checkpointing libraries for reproducibility analytics. However, capturing an entire checkpoint history for reproducibility study faces the challenges of high-frequency checkpointing across thousands of processes. As a result, the runtime overhead affects application performance and intermediate results when interleaving is introduced during floating-point calculations. In this paper, we extend asynchronous multi-level checkpoint/restart to study the intermediate results generated from scientific workflows. We present an initial prototype of a framework that captures, caches, and compares checkpoint histories from different runs of a scientific application executed using identical input files. We also study the impact of our proposed approach by evaluating the reproducibility of classical molecular dynamics simulations executed using the NWChem software. Experiment results show that our proposed solution improves the checkpoint write bandwidth when capturing checkpoints for reproducibility analysis by a minimum of 30× and up to 211× compared to the default checkpointing approach in NWChem.

KEYWORDS

result reproducibility, checkpoint analysis, high performance computing, asynchronous multi-level checkpointing

ACM Reference Format:
Kevin Assogba, Bogdan Nicolae, Hubertus Van Dam, and M. Mustafa Rafique. 2023. Asynchronous Multi-Level Checkpointing: An Enabler of Reproducibility using Checkpoint History Analytics. In *Workshops of The International Conference on High-Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3624062.3624256

1 INTRODUCTION

High-Performance Computing (HPC) applications produce massive amounts of distributed intermediate data during their execution that needs to be checkpointed concurrently by a large number of processes in real-time at scale. This is a fundamental I/O pattern used in a wide range of scenarios [23]: resilience based on checkpoint-restart, suspend-resume to continue long-running jobs over multiple reservations or to preempt batch jobs without losing progress in order to free resources for another [15], higher priority jobs producer-consumer patterns in workflows that communicate using intermediate data captured as checkpoints (e.g., simulations coupled with analytics), revisiting previous states to advance a computation (e.g., to enable out-of-core adjoint computations [16]), etc.

One of the key emerging scenarios enabled by checkpointing is the study of the *reproducibility* of scientific results. In this context, the increasing complexity of scientific applications and the extreme heterogeneity they face from all perspectives (different types of tasks, patterns, accelerators, job scheduling decisions, interleaving and competition for resources, etc.) makes it challenging to reason about reproducibility [7, 11]. A naive solution that simply compares the end results of two different application runs that start with the same input data does not enable enough insight. For example, if the end results are different, then there is no information available about what went wrong and when this happened during the runtime. Similarly, if there is a single valid path to reach the end result (which is often the case of HPC simulations), then obtaining a correct end result does not guarantee it was obtained through the valid path and not by coincidence through an alternative invalid path. Instead, checkpointing can be used to capture not only the end results of different application runs but also an entire history of intermediate checkpoints that describe the evolution of representative data structures during runtime. Thus, we can analyze the history of intermediate checkpoints to study reproducibility [1]. For example, we can compare each checkpoint of the history of one run with its equivalent from the history of another run to determine exactly when the two runs start diverging, what data structures were affected and how large the differences are. Similarly, we can check each checkpoint of the history against a set of invariants that describe a valid path to determine if the run has diverged from the valid path or not.

Capturing an entire history of checkpoints for the purpose of enabling the study of reproducibility is challenging for several reasons. First, the checkpoint frequency can be very high (e.g., a checkpoint may be needed at the end of every iteration of an HPC simulation) and may incur a significant runtime overhead due to I/O bottlenecks. The runtime overhead not only increases the time to solution and cost of obtaining the scientific results, but it may also affect the behavior of the application (e.g., it induces

SC-W 2023, November 12–17, 2023, Denver, CO, USA
2023. ACM ISBN 979-8-4007-0785-8/23/11…$15.00
https://doi.org/10.1145/3624062.3624256

different runtime decisions or interleavings under concurrency that affect intermediate results and/or cause deviation from the valid path). Thus, there is a need to minimize the overhead of checkpointing. Second, repeated runs of the same problem for the purpose of studying reproducibility do not always need to run to completion. For example, if the captured checkpoints of a second run show significant differences compared with the history of the first run early during the execution, it may be the case that enough information was already collected to enable a root cause analysis, in which case the second run can be terminated early to save time and resources.

In this paper, we propose a framework for reproducibility analytics based on the study of intermediate checkpoint histories that is specifically designed to address the aforementioned challenges. Our key idea is to extend asynchronous multi-level checkpointing techniques [20] with support to capture, cache, and compare histories of intermediate checkpoints of multiple application runs. Specifically, such techniques hide the overhead of checkpointing by blocking the application only for a minimal duration until a local copy of the checkpoint was captured on local storage (e.g., SSDs), from where it is flushed asynchronously to an external shared repository (e.g., parallel file system), which absorbs a majority of the I/O bottlenecks in the background. We summarize our contributions as follows:

- We illustrate how to study reproducibility based on checkpoint history analytics by considering the case of the northwest computational chemistry package (NWChem) [12], a widely used computational chemistry code (Section 2).
- We propose a series of design principles that leverage the fact that checkpoints are cached on fast local storage, which can be used to collectively optimize the flushing of the checkpoint history of multiple runs and accelerate comparisons. In particular, we insist on the need for a flexible analytics approach that enables both an offline comparison of the checkpoint histories, as well as an online comparison that enables early termination if desired (Section 3.1).
- We introduce an early prototype that provides a minimalist implementation of the design principles based on VELOC [20], a production-ready HPC checkpoint-restart library. In particular, VELOC offers versioning support for checkpoints, which we leverage to build a checkpoint history. Separately, we use an SQLite database instance to record additional metadata needed to compare the checkpoint histories of multiple runs (Section 3.2).
- We run a series of extensive experiments using a variety of NWChem scenarios. For each scenario, we study the runtime overhead introduced by frequent checkpointing necessary to capture the checkpoint history, and we characterize the evolution of key NWChem data structures from the checkpoint history across different runs, insisting on both quantitative and qualitative aspects (Section 4).

2 USE CASE: MOLECULAR DYNAMICS WITH NWCHEM

Molecular dynamics (MD) simulations study the behavior and properties of molecular systems over time. These simulations follow fundamental thermodynamics principles and equations (e.g., Newton's second law of motion) to explore changes in the state of molecules as they move, interact, and bind with other atoms over time in a molecular system. Software packages such as NWChem [13] and GAMESS [24] are often used to simulate computational workflows that model the desired molecular system.

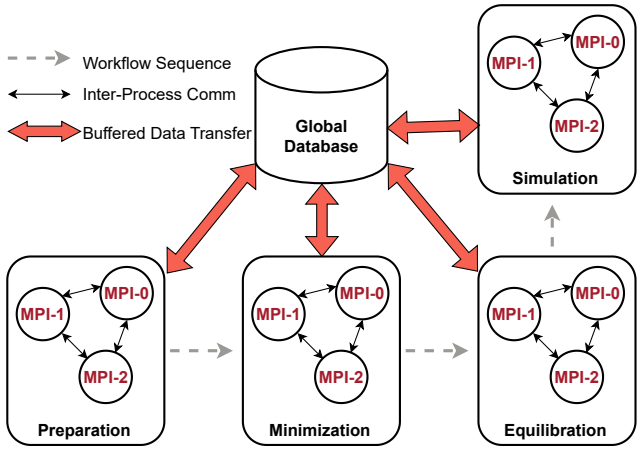


Figure 1: Illustration of a Molecular Dynamic Workflow

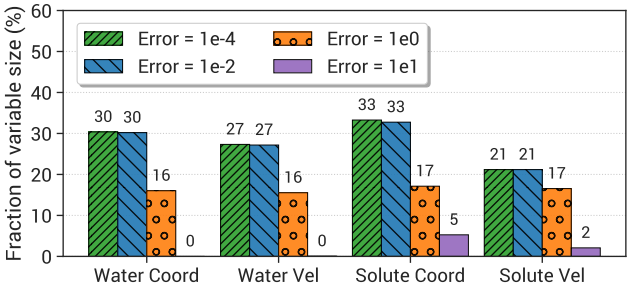
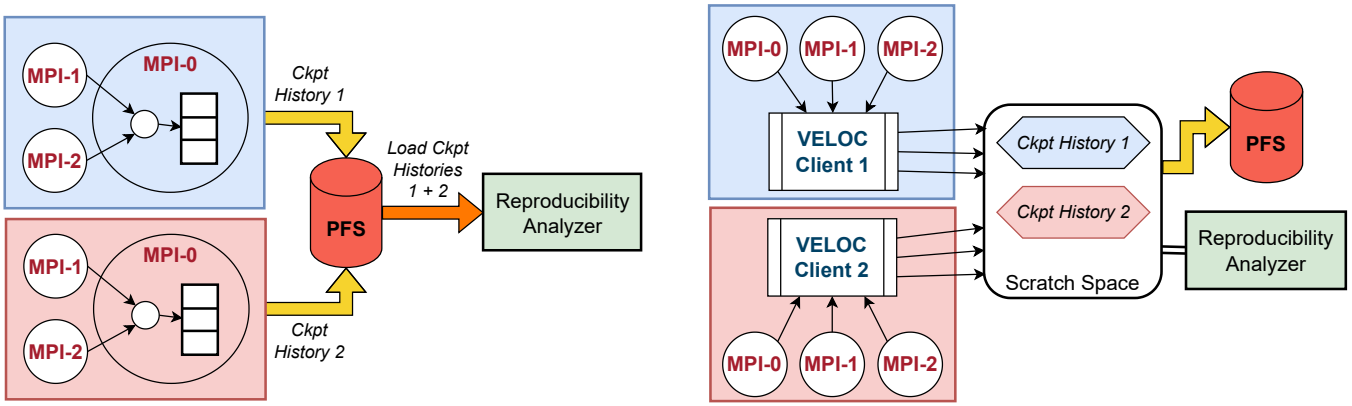


Figure 2: Illustration of the magnitude of errors induced by floating-point arithmetic in the Ethanol workflow.

In this paper, we focus on NWChem as a representative HPC application whose reproducibility can be studied by comparing the checkpoint history of two repeated runs. To this end, we choose a scenario that studies how protein atoms bind to the DNA, which we refer to as 1H9T [28].

Figure 1 illustrates the steps involved in this process. First, a preparation step is needed to read the structural information about the atoms in the system from a Protein Data Bank file (PDB) in order to generate a topology file and a restart file. The topology file contains static information about the system whereas the restart file captures dynamic information, and is regularly updated as the state of the system changes. Second, the topology file and a restart file are used to minimize atomic net forces and to write the new state back into the restart file. Third, a restrained equilibration step reorganizes and aligns the molecules in the system. Finally, an HPC simulation is responsible for solving the system.

These steps form an HPC workflow that involves multiple distributed processes, which coordinate through a global database



(a) Default NWChem **(b) VELOC + NWChem**

Figure 3: Intergration of VELOC with NWChem for checkpoint history analytics.

that provides a global view of the entire workflow for consistency. NWChem partitions the system into rectangular super-cells, allocates each cell to one process or rank, and uses the Global Array toolkit to enable each process to concurrently read, write, and update the region of the global variable that corresponds to their allocated cells.

For the purpose of this work, we focus on the equilibration step, which is critical in determining the outcome of the simulation. The equilibration uses an iterative technique that runs for a fixed number of iterations. After every K iteration, we capture several representative data structures (such as indices, coordinates, and velocities of water molecules and solute atoms) into a checkpoint on each process. The set of checkpoints corresponding to the same iteration is then added to the checkpoint history to capture the global evolution of the data structures on all distributed processes. The reproducibility analysis consists of comparing all checkpoints corresponding to the same iteration and the same process in the history of two repeated runs.

The captured data structures include a mix of integer values (e.g. indices) and floating point values (coordinates, velocities). Since floating point values are subject to numerical instability, simple byte-to-byte comparisons are not enough to determine equality. Even when applying an error threshold, the variability between different runs can be significant. For example, in Figure 2 we can observe that for the same data structure captured in the same checkpoint iteration, the differences between corresponding values of two repeated runs can cover a wide range (10^{-4} ... 10^{1}). While the explanations of this pattern are outside the scope of this work (and application-specific), this example illustrates that comparing two repeated runs is non-trivial, hence the need for scalable tools that facilitate reproducibility analytics.

3 PROPOSAL: REPRODUCIBILITY FRAMEWORK USING CHECKPOINT HISTORY ANALYTICS

3.1 Design Principles

Asynchronous Checkpoint Capture. Traditional checkpointing techniques are often *synchronous*, i.e., they block the application until the data structures have been captured and persisted into checkpoint files on a stable repository (typically a parallel file system). Although straightforward to implement, such techniques often cause an unacceptably high runtime overhead due to I/O bottlenecks, which are noticeable especially at scale when a large number of application processes compete for the limited I/O bandwidth of the repository where the checkpoints are stored. This runtime overhead is further exacerbated in the case reproducibility studies compared with traditional use cases of checkpointing (e.g. fault tolerance based on checkpoint-restart) for three reasons. First, the need to study the evolution of representative data structures at fine granularity leads to a higher checkpoint frequency. Second, reproducibility involves multiple repeated runs (at least two), each of which exhibits a high checkpoint frequency. Third, many HPC applications (such as NWChem) do not support distributed checkpointing, collecting instead the data structures on a single process that is responsible for writing the checkpoint (illustrated in Figure 3a). If the runtime overhead due to checkpointing is high, this may trigger a different application behavior (e.g., different interleaving of parallel patterns) that potentially affects the evolution of the intermediate data structures between different runs. In this case, checkpointing is an invasive operation. Even when checkpointing is not an invasive operation, a high runtime overhead accumulates quickly over multiple runs and results in extra costs and/or missed deadlines. To mitigate this issue, multi-level checkpointing strategies (as illustrated by VELOC [19, 20] and SCR [17]) can be used that leverage hierarchic node-local storage tiers (e.g., GPU memory, host memory, non-volatile memory, emerging CXL-based memory [3], and SSDs) to flush the checkpoints to the slower tiers asynchronously. Specifically, in this case, the application is blocked only for the duration of the writes to the fastest tier, while other

cascading transfers to the slower tier proceed concurrently in the background. This is illustrated in Figure 3b. We argue that such techniques are highly effective and propose to extend them for the purpose of reproducibility, which we discuss next.

Flexible Offline/Online Analytics. In the simplest form, the study of reproducibility involves two repeated runs, each of which independently captures and persists the intermediate checkpoint history to a parallel file system using asynchronous multi-level checkpointing techniques. In this case, the reproducibility analysis is decoupled from the actual runs and can be performed at any later moment in *offline* fashion. However, doing so may unnecessarily execute the second run to completion even if a divergence between the two runs can be observed early in the checkpoint history, which may be enough to identify a root cause and therefore justify an early termination. Thus, we argue in favor of a flexible solution that enables an alternative *online* reproducibility analytics that consumes the checkpoint history of the first run on the fly while the second run progresses. Specifically, as soon as a checkpoint corresponding to the same process and iteration is available for both the first and second runs, a comparison can be made asynchronously without blocking the progress of either run. Then, if the checkpoints are considered divergent, early termination can be triggered. Such an approach is particularly effective in combination with asynchronous multi-level checkpointing techniques because the comparisons can be introduced in the asynchronous I/O pipeline in order to minimize the runtime overheads. Optimizations in this direction will be discussed next.

Cache and Reuse Checkpoint History on Local Storage. A naive implementation may simply perform repeated reads from the parallel file system to compare all checkpoints corresponding to the same processes and the same iteration from the two checkpoint histories. However, such a naive approach has an important disadvantage: it incurs high overheads due to the need to read large amounts of data from the parallel file system under concurrency, which often leads to I/O bottlenecks. As a consequence, it is not enough to optimize asynchronous multi-level checkpointing techniques just to write checkpoints efficiently (which is often sufficient for scenarios like fault tolerance, since reading checkpoints is comparatively infrequent). Instead, we need to co-optimize the problem of writing and revisiting checkpoints later. To this end, we propose to extend approaches such as those proposed in [16], which employ multi-level access pattern aware caching and prefetching techniques in order to anticipate and accelerate the full cycle of writing and reading a checkpoint history. In this context, there are several opportunities. First, the buffers reserved for caching and prefetching on different storage tiers can be shared by multiple runs. For example, if offline analytics is desired, it is not necessary to wait for the first run to finish before starting the second run. In fact, both runs can be started simultaneously at the expense of write competition between the two runs for the storage tiers used by multi-level caching and prefetching techniques. If online analytics is desired, the problem is further complicated by the interleaving of reads and writes belonging to different runs. Thus, our proposed extensions aim to mitigate the interference between different runs in order to maximize the reuse of checkpoints on the fastest storage tiers. Second, even if the checkpoints are read from the fastest storage tiers, the overhead of comparing large checkpoints iterating over their whole content can be significant. As a consequence, we envision novel comparison techniques that are based on hierarchic hashing (similar to Merkle trees) and are tolerant to floating point variations (as discussed in Section 2). Such an approach only needs to revisit hashing metadata instead of the full checkpoint pairs, at the expense of additional computational overhead needed for hashing. To mitigate the computational overhead, we extend high-performance hashing techniques originally used for de-duplicating checkpointing data [25].

Algorithm 1: VELOC Checkpointing for NWChem.

```
   Input: ckptname, conf_file, ftn_arrs, step, maxstep, err
1  begin
2      if step == 0 then
3          ga_comm ←
             ga_mpi_comm_pgroup_default(ga_comm)
4          VELOC_Init(ga_comm, conf_file, err)
5      for each ftn_arr in ftn_arrs do
6          c_arr ← Transpose(ftn_arr)
7          VELOC_Mem_protect (step, c_loc(c_arr),
8                  size(c_arr), sizeof(c_arr[1]), err)
9          VELOC_Checkpoint(ckptname, step, err)
10     if step == maxstep then
11         VELOC_Finalize(err)
```

3.2 Implementation

In this section, we introduce a preliminary implementation that integrates VELOC with NWChem for the purpose of evaluating the potential benefits of the design principles discussed in Section 3. This implementation can be easily adapted to other HPC applications that are capable of checkpointing intermediate data.

NWChem supports diverse chemistry simulation workflows including classical MD, density functional theory, etc. Since NWChem is implemented in Fortran, we had to develop Fortran bindings for VELOC, which in turn call its C++ API. Our integration with NWChem requires solving two main challenges: the representation of array data structures and the capture of checkpoint descriptors (the name of the workflow, checkpoint iteration, the process ID, and data types and dimensions of variables to checkpoint).

Checkpoint Data Representation. Fortran and C++ represent arrays in different ways. Fortran stores arrays in column-major order while C++ uses a row-major order. As a consequence, it is not enough to simply pass pointers to Fortran arrays into C++. We had to implement a transposition function in the comparison pipeline that converts the column-major order to the row-major order.

Checkpoint Annotation. By default, VELOC captures the checkpoint name, the process ID, and a user-defined version number (typically the iteration number) into the checkpoint history. Furthermore, each checkpoint file contains a header that describes the captured data structures and their sizes. However, the header does not contain information about the type of the data structures. This

Table 1: Summary of checkpointing and comparison time on 1H9T, Ethanol and Ethanol-4 workflows.

Workflow	# of Ranks	Ckpt time (ms)		Ckpt size (KB)		Comparison time (ms)	
		Our Solution	Default	Our Solution	Default	Our Solution	Default
1H9T	4	1.96	49.46	1480	1356	602	605
1H9T	8	1.01	47.12	1492	1356	834	837
1H9T	16	0.65	48.83	1524	1356	1352	1358
Ethanol	4	0.45	7.55	52	96	583	586
Ethanol	8	0.31	8.67	64	96	862	866
Ethanol	16	0.50	10.78	68	96	1306	1312
Ethanol-4	4	0.82	141.81	2972	4764	591	594
Ethanol-4	8	0.69	148.25	2984	4764	886	889
Ethanol-4	16	0.62	154.19	3004	4764	1361	1365

missing information is important in the context of reproducibility because it determines how the comparison must be performed (i.e., exact for integers and approximate for floating points). Therefore, we collect additional information about the type of the data structures and annotate the checkpoints accordingly.

Checkpoint History Capture. We asynchronously checkpoint the application data using VELOC's API as detailed in Listing 1. To align with NWChem, we intersect the global MPI communicator used by the application to initialize the VELOC client. We use the VELOC_Mem_protect API to declare the memory regions that are part of the checkpoint and serialize such regions as a new checkpoint using the VELOC_Checkpoint API. Each checkpoint has a corresponding ID that we define to be the number of the current simulation step. Our implementation focuses on two-level checkpointing using one temporary scratch space (local storage available on the compute nodes) and one persistent repository (a parallel file system). We defined our checkpointing frequency to match the frequency of rewriting the NWChem restart file. This restart frequency is pre-defined in the NWChem input file, therefore, we do not require users to explicitly define a checkpointing frequency parameter.

Analysis of Checkpoint Histories. To analyze collected checkpoints, we restore the checkpoint histories into the host memory using the VELOC_Restart API. Our current prototype implements two types of comparison: exact and approximate comparison of the checkpoints. Each checkpoint in our use case application (classical MD) captures the indices, coordinates, and velocities of water molecules and solute atoms. NWChem uses the C++ equivalent of a 64-bit integer for indices, and double-precision floating-point numbers for coordinates and velocities. We apply the exact comparison to compare the indices because they are whole numbers and perform an approximate comparison of coordinates and velocities. Floating-point numbers are non-associative and non-distributive and, thereby, are subject to rounding errors that propagate from one MD calculation to the other. These properties render an exact comparison impractical and require an approximate comparison with respect to an error margin ϵ. We compare two floating-point numbers a and b by verifying whether $|a - b| > \epsilon$. If the condition is satisfied, we label the corresponding index as a *mismatch*.

4 PRELIMINARY EVALUATION

4.1 Setup

We evaluate our approach on the Polaris HPC system, an Argonne Leadership Computing Facility (ALCF) system that consists of 560 compute nodes. Each node is equipped with a single 2.8 GHz AMD EPYC Milan CPU (32-cores), 512 GiB DDR4 memory, and 4 NVIDIA A100 GPUs. The nodes can access a 10 TB Lustre parallel file system through a POSIX mount point. While our experiments did not require access to a GPU, this HPC system setup regularly hosts large-scale scientific applications. For our evaluation, we leverage a single node on the system to execute our selected MD workflows (Section 4.2). In terms of the software ecosystem, we used VELOC 1.5 and NWChem software with dependencies such as GFortran 11.2, GCC/G++ 11.2, and MPICH 8.1.

4.2 Workflows

We base our evaluation on two popular molecular dynamics workflows: 1H9T [28] and Ethanol [29]. 1H9T workflow studies the transcription of genes focusing on the binding process between a protein and DNA. This MD simulation executes a sequence of steps including a system preparation, minimization calculation, equilibration calculation, and molecular dynamics simulation. Our evaluation focuses on the equilibration calculation which is the last step before the MD simulation. We aim to study the reproducibility of the intermediate results that are used during the simulation. The Ethanol workflow simulates the dynamics of a single ethanol molecule in water. We add three variants of this workflow to evaluate the strong and weak scalability of our proposed approach: Ethanol-2, Ethanol-3, and Ethanol-4. These variants increase the number of unit cells per supercell in the molecular system, requiring

respectively 8×, 27×, and 64× the number of processes allocated to the base ethanol simulation for weak scalability. We execute each workflow for 100 iterations and collect a checkpoint every 10 iterations.

4.3 Compared Approaches

We compare the following two approaches using the offline comparison method:

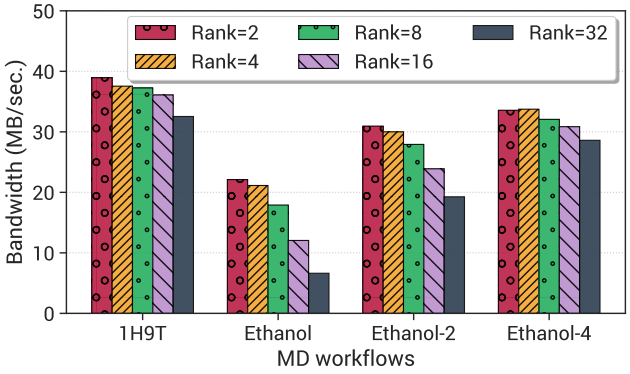
Default NWChem. This is the default strategy used in NWChem: the data processed by each MPI rank is gathered on one process and synchronously flushed to the PFS. We use a Lustre filesystem to store the checkpoint histories generated by this approach and reload the histories of the first and second execution of the workflow back into the host memory after the completion of both executions.

Our Approach. Our approach uses asynchronous multi-level checkpointing to capture workflow checkpoint histories and compares the critical data contained in the checkpoints of two independent executions of the workflow to study the workflow's reproducibility. We use a temporary memory-based filesystem (TMPFS) as scratch storage and the PFS as persistent storage.

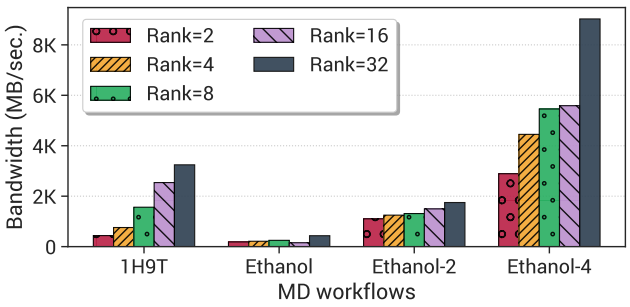
4.4 Results

Impact of Asynchronous Multi-level Checkpointing on Reproducibility Analysis. We first analyze the checkpointing and comparison overheads for the workflows described in Section 4.3. Table 1 summarizes the time spent by the compared approaches to capture and compare the checkpoint histories of two repeated runs. We observe that our approach improves the checkpointing time by a minimum of 30× and up to 211× compared to Default NWChem. This benefit is due to asynchronous checkpointing where VELOC first stages the checkpoints on TMPFS before flushing to the PFS. Meanwhile, the default approach takes longer because only one process synchronously flushes the checkpoints to the PFS. At the end of the second execution, the checkpoint histories are loaded back into the host memory for comparison. Such data transfer from PFS to host memory also increases the time to compare checkpoint histories as opposed to VELOC which directly loads from TMPFS into the host memory.

Strong scalability of checkpointing throughput. Next, we compare the checkpointing throughput of default NWChem with our solution. First, we focus our study on strong scalability: we increase the number of MPI ranks used to execute the MD simulation while keeping the number of cells in the molecular system fixed. We study four different configurations. For each of these configurations, we measure the checkpointing throughput (bandwidth), which results in comparable quantities despite different checkpoint sizes. Figure 4 shows that Default NWChem achieves a peak write bandwidth of 39 MB/s when executing the 1H9T workflow using 2 MPI ranks whereas our solution yields 8.8 GB/s using 32 MPI ranks on the Ethanol-4 workflow. With the same checkpoint size, as shown in Table 1, the performance of Default NWChem reduces as more processes are used to execute the simulation because the main MPI rank spends an increasing amount of time gathering the same data size from all the ranks. This performance reduction also



(a) Default NWChem checkpoint write bandwidth



(b) VELOC checkpoint write bandwidth

Figure 4: Analysis of checkpoint write bandwidth.

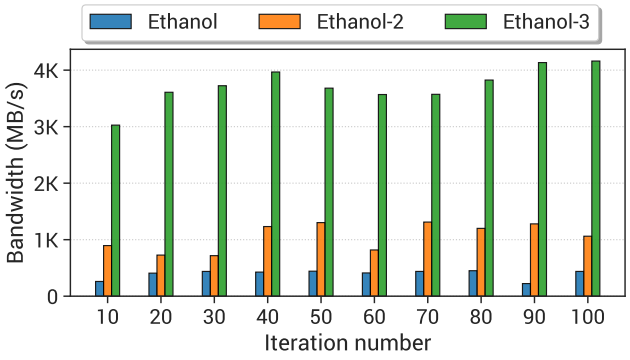


Figure 5: Weak scalability analysis of VELOC checkpointing using the Ethanol workflow variants.

highlights a disadvantage of using a single process for synchronous checkpointing. On the other hand, an asynchronous multi-level checkpointing solution is both faster and more scalable, since all application processes contribute to saving their checkpoints concurrently with VELOC. Consequently, despite the small increase in the checkpoint size (due to additional metadata attached to the checkpoint of each MPI rank), the increased number of processes improves the write bandwidth.

Weak scalability of checkpoint throughput. Figure 5 reports on the performance benefits of using VELOC to capture checkpoints

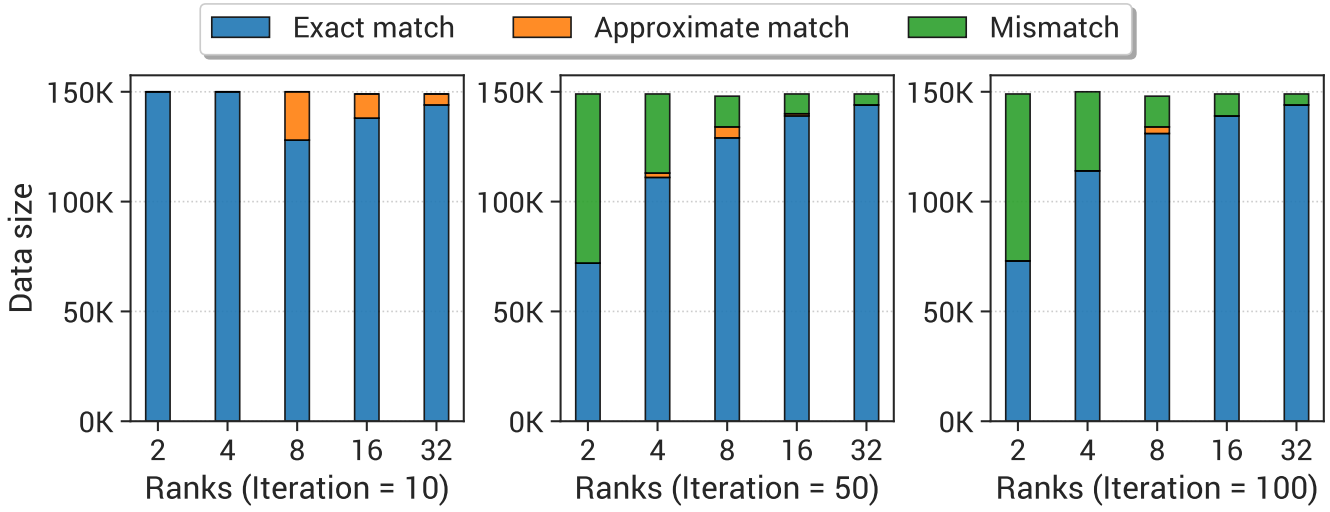


Figure 6: Comparison of the velocities of water molecules from two executions of the Ethanol-4 workflow.

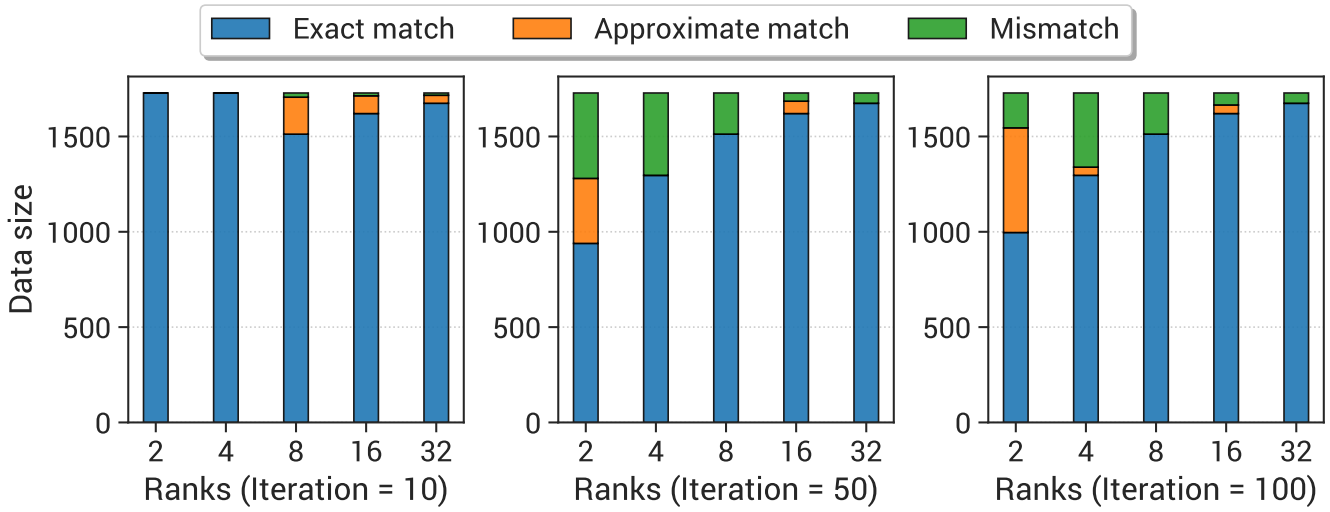


Figure 7: Comparison of the velocities of solute atoms from two executions of the Ethanol-4 workflow.

in weak scaling scenarios. We design weak scalability experiments by executing Ethanol, Ethanol-2, and Ethanol-3 workflows with 1, 8, and 27 MPI ranks, respectively. We observe that our solution maintains a peak bandwidth of up to 4 GB/s in weak scaling scenarios. The maximum bandwidth reduces by ≈2× compared to strong scaling scenarios described in Figure 4 because of the increased interference and contention for I/O resources. The increase in the number of MPI ranks does not alleviate the workload of each rank as the workload size increases at the same rate, and each rank continues to process the same number of cells in the molecular system. Nevertheless, our solution yields up to a 5× increase in bandwidth during the Ethanol-2 simulation with respect to the Ethanol simulation. The same scale of bandwidth increase is observed when executing the Ethanol-3 simulation.

Checkpoint history comparison. Our final set of experiments focuses on the analysis of collected checkpoint histories. We evaluate the number of mismatches in compared checkpoints using two types of comparisons: exact and approximate. The exact comparison is applied for integer values and checks whether they are equal in their binary representations, whereas the approximate comparison is applied for floating point values and evaluates whether the absolute difference between them is lower than an error value. For our evaluation, we use an error of 1e-4. This value is selected based on previous NWChem studies [30], in which a relative error of 1e-4 was observed as a result of bit-flips in the representation of double-precision floating point numbers.

Figures 6 and 7 respectively report the number of exact matches, approximate matches, and mismatches in the velocities of water

molecules and solute atoms from the Ethanol-4 simulation. We present the comparison of the first checkpoint (iteration 10), the fifth checkpoint (iteration 50), and the last checkpoint (iteration 100). We observe that the rounding error from floating point calculations accumulates across iterations. For example, we notice such a trend in Figures 6 and 7 where the first checkpoints with 2 and 4 MPI ranks yield no mismatch, but, the error accumulates during the following iterations leading to more approximate matches and mismatches at the fifth checkpoint iteration. This instability in floating-point numbers can also lead to reduced error, as with the velocities of the solute atoms in the last checkpoint where some mismatches in the fifth checkpoint qualify as approximate matches. These variations highlight the importance of considering an entire checkpoint history for reproducibility analytics.

5 RELATED WORK

The reproducibility of computational workflows is a principle for scientific integrity and reusability. When a scientific application executes multiple times using identical input files and source code on the same computing infrastructure, reproducibility entails that the application follows the same execution path, i.e., identical traces of function calls from the first instruction to the last, produces the same intermediate results, and yields the same outcome. The scientific community agrees on the prevalence of reproducibility, but executing applications on high-performance computers challenges the guarantee of reproducibility, e.g., due to dynamic scheduling of parallel processes [2]. A small variation in the computing environment, e.g., a different version of a software dependency or adding an intermediate calculation step in the workflow, creates numerical instability and yields computational irreproducibility [26]. Variability in the computing environment, e.g., network bandwidth is also a cause for non-reproducible application performance [27].

For resilience and fault tolerance, HPC applications have adopted the checkpoint-restart model to recover from application failures without losing computational results produced before the failure. Multi-level asynchronous I/O frameworks for checkpointing, e.g., VELOC[20] and ADIOS2 [9], relieve the overhead of flushing application checkpoints at high frequency to the PFS. Asynchronous checkpointing captures intermediate results with minimal overhead on the application [10], enabling the study of intermediate checkpoints to analyze the application's reproducibility before its completion.

To reduce the effect of interference from other applications on reproducibility, workflow management systems [8] are increasingly adopting sandboxed environments, e.g., docker and singularity containers, to isolate computations [5, 6] and enable a continuous analysis of the computational workflow with minimal startup cost and complexity [4]. This solution alleviates external effects but fails to account for errors related to the target applications. The current I/O pattern of scientific applications to flush checkpoints directly to the PFS presents a risk of interleaves that affect ongoing computations and introduce varying errors on intermediate results. Existing studies explore strategies to improve reproducibility by reducing numerical roundoff errors introduced with floating-point arithmetic [2, 14], e.g., error-free transformation for reproducible summation [18], but as the scale of execution environment and the complexity of data types used during computations increase, current solutions will need to be revisited. This paper advocates analytical tools to identify the root causes of irreproducibility.

Reproducibility analytics usually focuses on workflow provenance [21] and performance data, e.g., execution time [22], but little attention is given to the comparison of intermediate results produced by the function calls and steps along the provenance graph. These comparisons can contribute to identifying the root causes of the application's irreproducibility. In this paper, we leverage the checkpoint-restart mechanism, already used by scientific applications, to instrument the analysis of intermediate results reproducibility.

6 CONCLUSION

In this work, we propose a framework for reproducibility analytics that is based on the idea of comparing histories of intermediate checkpoints captured during repeated runs of HPC applications. We envision a solution based on several design principles: adapting asynchronous multi-level checkpointing techniques in the context of reproducibility, a flexible approach to compare the checkpoint histories in both offline and online fashion, as well as scalable techniques for caching, prefetching and hash-based comparison that maximize the use of fast cache tiers and, respectively, minimize the comparison overheads. To this end, we implemented an early prototype based on VELOC, a production-level asynchronous multi-level checkpointing library, and integrated our solution with NWChem, a computational chemistry software, to study the reproducibility of HPC molecular dynamics simulation workflows. Preliminary evaluations show that our solution has promising potential to reduce the checkpointing overhead compared with the default checkpointing strategy employed by NWChem: it is up to 211× faster with a minimum checkpoint bandwidth increase of 30×. This is important not only to save core hours and reduce the runtime and resource utilization but also to reduce the influence of checkpointing on the behavior of the application. Furthermore, we observed interesting patterns in the evolution of the checkpointed data between two runs: some of the variables are stable while others begin to diverge and increase their divergence over time. Encouraged by these preliminary results, we plan to explore in future work several promising avenues: (1) interpret the results with domain-specific knowledge to explain the observed divergence; (2) design, develop, and implement novel algorithms and techniques based on the design principles and demonstrate their effectiveness in speeding up the reproducibility analytics; (3) demonstrate the effectiveness of these algorithms at scale compared with the preliminary implementation introduced in this work, as well as compared with other state-of-art techniques.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under grants 0269227 and 220807, as well as under contract DE-AC02-06CH11357. Furthermore, it was supported by the National Science Foundation (NSF) under Awards No. 2106634 and 2106635.

REFERENCES

[1] Ishan Abhinit, Emily K. Adams, Khairul Alam, Brian Chase, Ewa Deelman, Lev Gorenstein, Stephen Hudson, Tanzima Islam, Jeffrey Larson, Geoffrey Lentner, Anirban Mandal, John-Luke Navarro, Bogdan Nicolae, Line Pouchard, Rob Ross, Banani Roy, Mats Rynge, Alexander Serebrenik, Karan Vahi, Stefan Wild, Yufeng Xin, Rafael Ferreira da Silva, and Rosa Filgueira. 2022. Novel Proposals for FAIR, Automated, Recommendable, and Robust Workflows. In *Proceedings of the 17th Workshop on Workflows in Support of Large-Scale Science (WORKS'22 in conjunction with SC'22)* (Dallas, USA). IEEE Computer Society, Los Alamitos, CA, USA, 84 – 92.

[2] Peter Ahrens, James Demmel, and Hong Diep Nguyen. 2020. Algorithms for efficient reproducible floating point summation. *ACM Transactions on Mathematical Software (TOMS)* 46, 3 (2020), 1–49.

[3] Moiz Arif, Kevin Assogba, M. Mustafa Rafique, and Sudharshan Vazhkudai. 2023. Exploiting CXL-Based Memory for Distributed Deep Learning. In *Proceedings of the 51st International Conference on Parallel Processing* (Bordeaux, France) *(ICPP '22)*. Association for Computing Machinery, New York, NY, USA, Article 19, 11 pages. https://doi.org/10.1145/3545008.3545054

[4] Brett K Beaulieu-Jones and Casey S Greene. 2017. Reproducibility of computational workflows is automated using continuous analysis. *Nature biotechnology* 35, 4 (2017), 342–346.

[5] R. Shane Canon. 2020. The Role of Containers in Reproducibility. In *Proceedings of the 2020 2nd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. IEEE Computer Society, Los Alamitos, CA, USA, 19–25. https://doi.org/10.1109/CANOPIEHPC51917.2020.00008

[6] Shreyas Cholia, Lindsey Heagy, Matthew Henderson, Drew Paine, Jon Hays, Ludovico Bianchi, Devarshi Ghoshal, Fernando Pérez, and Lavanya Ramakrishnan. 2020. Towards Interactive, Reproducible Analytics at Scale on HPC Systems. In *Proceedings of the 2020 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*. IEEE Computer Society, Los Alamitos, CA, USA, 47–54. https://doi.org/10.1109/UrgentHPC51945.2020.00011

[7] Peter V Coveney, Derek Groen, and Alfons G Hoekstra. 2021. Reliability and reproducibility in computational science: Implementing validation, verification and uncertainty quantification in silico. *Philosophical Transactions of the Royal Society A* 379, 2197 (2021), 20200409.

[8] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. 2017. Nextflow enables reproducible computational workflows. *Nature biotechnology* 35, 4 (2017), 316–319.

[9] William F Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, et al. 2020. Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX* 12 (2020), 100561.

[10] Mikaila J Gossman, Bogdan Nicolae, Jon C Calhoun, Franck Cappello, and Melissa C Smith. 2021. Towards aggregated asynchronous checkpointing. *arXiv preprint arXiv:2112.02289* (2021).

[11] Bin Hu, Shane Canon, Emiley A Eloe-Fadrosh, Michal Babinski, Yuri Corilo, Karen Davenport, William D Duncan, Kjiersten Fagnan, Mark Flynn, Brian Foster, et al. 2022. Challenges in bioinformatics workflows for processing microbiome omics data at scale. *Frontiers in Bioinformatics* 1 (2022), 826370.

[12] Ricky A. Kendall, Edoardo Aprà, David E. Bernholdt, Eric J. Bylaska, Michel Dupuis, George I. Fann, Robert J. Harrison, Jialin Ju, Jeffrey A. Nichols, Jarek Nieplocha, T.P. Straatsma, Theresa L. Windus, and Adrian T. Wong. 2000. High performance computational chemistry: An overview of NWChem a distributed parallel application. *Computer Physics Communications* 128, 1 (2000), 260–283.

[13] Karol Kowalski, Raymond Bair, Nicholas P Bauman, Jeffery S Boschen, Eric J Bylaska, Jeff Daily, Wibe A de Jong, Thom Dunning Jr, Niranjan Govind, Robert J Harrison, et al. 2021. From NWChem to NWChemEx: Evolving with the computational chemistry landscape. *Chemical reviews* 121, 8 (2021), 4962–4998.

[14] Kuan Li, Kang He, Stef Graillat, Hao Jiang, Tongxiang Gu, and Jie Liu. 2023. Multi-level parallel multi-layer block reproducible summation algorithm. *Parallel Computing* 115 (2023), 102996.

[15] Avinash Maurya, Bogdan Nicolae, Ishan Guliani, and M. Mustafa Rafique. 2021. CoSim: A Simulator for Co-Scheduling of Batch and on-Demand Jobs in HPC Datacenters. In *Proceedings of the IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications* (Prague, Czech Republic) *(DS-RT '20)*. IEEE Press, 167–174.

[16] Avinash Maurya, M. Mustafa Rafique, Thierry Tonellot, Hussain J. AlSalem, Franck Cappello, and Bogdan Nicolae. 2023. GPU-Enabled Asynchronous Multi-Level Checkpoint Caching and Prefetching. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing* (Orlando, FL, USA) *(HPDC '23)*. Association for Computing Machinery, New York, NY, USA, 73–85. https://doi.org/10.1145/3588195.3592987

[17] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. 2010. Design, Modeling, and Evaluation of a Scalable Multi-Level Checkpointing System. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10)*. IEEE Computer Society, USA, 1–11. https://doi.org/10.1109/SC.2010.18

[18] Ingo Müller, Andrea Arteaga, Torsten Hoefler, and Gustavo Alonso. 2018. Reproducible Floating-Point Aggregation in RDBMSs. In *Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1049–1060. https://doi.org/10.1109/ICDE.2018.00098

[19] Bogdan Nicolae, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Franck Cappello. 2019. VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale. In *Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, Los Alamitos, CA, USA, 911–920. https://doi.org/10.1109/IPDPS.2019.00099

[20] Bogdan Nicolae, Adam Moody, Greg Kosinovsky, Kathryn Mohror, and Franck Cappello. 2021. VELOC: VEry Low Overhead Checkpointing in the Age of Exascale. In *Proceedings of the First International Symposium on Checkpointing for Supercomputing (SuperCheck'21)*. Virtual Event.

[21] Line Pouchard, Sterling Baldwin, Todd Elsethagen, Shantenu Jha, Bibi Raju, Eric Stephan, Li Tang, and Kerstin Kleese Van Dam. 2019. Computational reproducibility of scientific workflows at extreme scales. *The International Journal of High Performance Computing Applications* 33, 5 (2019), 763–776.

[22] Srinivasan Ramesh, Mikhail Titov, Matteo Turilli, Shantenu Jha, and Allen Malony. 2022. The Ghost of Performance Reproducibility Past. In *Proceedings of the 2022 IEEE 18th International Conference on e-Science (e-Science)*. IEEE Computer Society, Los Alamitos, CA, USA, 513–518. https://doi.org/10.1109/eScience55777.2022.00091

[23] Robert Ross, Lee Ward, Philip Carns, Gary Grider, Scott Klasky, Quincey Koziol, Glenn K. Lockwood, Kathryn Mohror, Bradley Settlemyer, and Matthew Wolf. 2018. *Storage Systems and Input/Output: Organizing, Storing, and Accessing Data for Scientific Discovery. Report for the DOE ASCR Workshop on Storage Systems and I/O. [Full Workshop Report]*. Technical Report DOE/SC-0196. US Department of Energy. Conference: Storage Systems and I/O: Organizing, Storing, and Accessing Data for Scientific Discovery, Gaithersburg, MD, 19-20 Sep 2018.

[24] Michael W. Schmidt, Kim K. Baldridge, Jerry A. Boatz, Steven T. Elbert, Mark S. Gordon, Jan H. Jensen, Shiro Koseki, Nikita Matsunaga, Kiet A. Nguyen, Shujun Su, Theresa L. Windus, Michel Dupuis, and John A. Montgomery Jr. 1993. General atomic and molecular electronic structure system. *Journal of Computational Chemistry* 14, 11 (1993), 1347–1363.

[25] Nigel Tan, Jakob Lüttgau, Jack Marquez, Keita Teranishi, Nicolas Morales, Sanjukta Bhowmick, Franck Cappello, Michela Taufer, and Bogdan Nicolae. 2023. Scalable Incremental Checkpointing using GPU-Accelerated De-Duplication. In *Proceedings of the 52nd International Conference on Parallel Processing, ICPP 2023, Salt Lake City, UT, USA, August 7-10, 2023*. ACM, 665–674. https://doi.org/10.1145/3605573.3605639

[26] Krishna Tiwari, Sarubini Kananathan, Matthew G Roberts, Johannes P Meyer, Mohammad Umer Sharif Shohan, Ashley Xavier, Matthieu Maire, Ahmad Zyoud, Jinghao Men, Szeyi Ng, et al. 2021. Reproducibility in systems biology modelling. *Molecular systems biology* 17, 2 (2021), e9982.

[27] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. 2020. Is Big Data Performance Reproducible in Modern Cloud Networks?. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation* (Santa Clara, CA, USA) *(NSDI'20)*. USENIX Association, USA, 513–528.

[28] Daan MF Van Aalten, Concetta C DiRusso, and Jens Knudsen. 2001. The structural basis of acyl coenzyme A-dependent regulation of the transcription factor FadR. *The EMBO journal* 20, 8 (2001), 2041–2050.

[29] Hubertus JJ van Dam. [n. d.]. Ethanol Test Case. https://github.com/hjjvandam/nwchem-1/tree/pretauadio2/QA/tests/ethanol. Accessed: 2023-08-18.

[30] Hubertus JJ van Dam, Abhinav Vishnu, and Wibe A De Jong. 2013. A case for soft error detection and correction in computational chemistry. *Journal of Chemical Theory and Computation* 9, 9 (2013), 3995–4005.