# Performance Impact of I/O on QMCPack Simulations at the Petascale and Beyond

S. Herbein, M. Matheny, M. Wezowicz
University of Delaware
Newark, DE 19716

J. Krogel, J. Logan, J. Kim, S. Klasky
Oak Ridge National Laboratory
Oak Ridge, TN 37831

M. Taufer
University of Delaware
Newark, DE 19716

***Abstract*—Traditional petascale applications, such as QMCPack, can scale their computations to completely utilize modern supercomputers like Titan, but they cannot scale their I/O. To preserve scalability, scientists cannot save data at the granularity needed to enable scientific discovery and are forced to use large intervals between two checkpoint calls. In this paper, we work to increase the granularity of the I/O in QMCPack simulations without increasing the I/O associated overhead or compromising the scalability of the simulations. Our solution redesigns the I/O algorithms used by QMCPack to gather finer-grained data at high frequencies and integrate the ADIOS API to select effective I/O methods without major code changes. The extension of a tool such as Skel to mimic the variable I/O in QMCPack allows us to predict the I/O performance of the code when using ADIOS methods at the petascale. We show how I/O libraries like ADIOS allow us to increase the amount of scientific data extracted from QMCPack simulations at the granularity desired by the scientists while keeping the I/O overhead below 10%. We also show how the impact of checkpoint I/O for the QMCPack code using ADIOS is below 5% when using preventive tactics for checkpointing at the petascale and beyond.**

## I. Introduction

As supercomputers and their applications begin to move closer to exascale, scientists will have to sacrifice writing fine-grained scientific data to disk to maintain simulation scalability. The Quantum Monte Carlo code, QMCPack, is already facing this challenge at the petascale. QMCPack's existing I/O methods cannot write fine-grained data at scale without compromising scalability. As a result, averaged statistical values across the entire simulated molecular system, rather than the exact values for each particle, are written to disk. For the same scalability reason, the checkpoint data needed to restart a crashed simulation is written to disk only after large time intervals. Preliminary tests showed that even for QMCPack simulations of simple molecular systems, the time spent in I/O associated with checkpointing might contribute to a large portion of the total wall-clock time. Figure 1 shows an example of the time spent on execution (computation and communication) versus the time spent on I/O checkpointing for a QMCPack simulation of simple water molecules for a fine-grained checkpoint frequency. When using 64 nodes on a small cluster at Oak Ridge National Laboratory, close to 90% of the execution time is spent writing to disk. Overall, similar simulations of larger systems on a larger number of nodes can become prohibitively long.

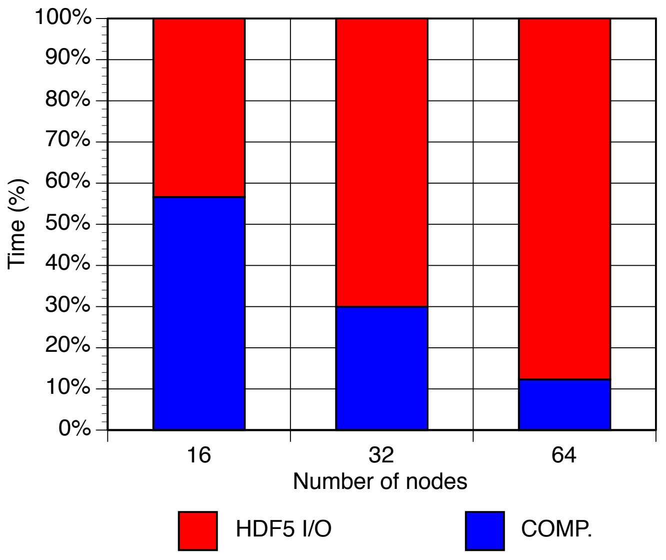


Fig. 1: Time distribution in terms of execution (i.e., compute and communication) and I/O for a water simulation with QMCPack and HDF5.

The goal of the work in this paper is to present solutions that allow users to increase both the granularity of scientific and checkpoint data written to disk in QMCPack simulations without increasing I/O overhead or compromising the simulation's scalability. The contributions of this paper are twofold:

- We extend QMCPack code to efficiently support fine-grained writing of scientific data to disk. To do so, we first redesign the I/O algorithms of QMCPack to write simulation data at a fine-grained per-process level (i.e., positions and energies of each single particle) rather than at a coarse-grained per-simulation level (i.e., average energies of the whole molecule). Then we reengineer the original and extended QMCPack codes to use the ADIOS (Adaptable I/O System) I/O library rather than HDF5 to write in output positions and energies at both fine-grained and coarse-grained levels. We analyze the performance of QMCPack writing in output fine-grained and coarse-grained data by using HDF5 and ADIOS.

- We assess the impact of I/O on checkpointing for the QMCPack code using ADIOS on extreme-scale platforms. To do so, we first redesign the checkpoint algorithms of QMCPack to use ADIOS rather than HDF5 for writing to disk the particle positions needed to restart crashed simulations. We also extend the Skel tool to emulate the variable I/O in QMCPack during checkpointing and use the tool to collect a large set of I/O times without running entire simulations. We then apply regression and extrapolation methods to the time dataset generated by a combination of QMCPack and Skel runs to estimate the checkpoint times at a large scale. We use the estimated times to predict the impacts of I/O on QMCPack simulation times on extreme-scale systems when preventive and proactive tactics for checkpointing are used.

The rest of this paper is organized as follows: Section II provides a brief overview of the software used in this work and the related work; Section III describes the new versions of the QMCPack code that use ADIOS to write fine-grained statistics and checkpoint data as well as the extended Skel tool to estimate the I/O performance of QMCPack; Section IV presents the performance analysis and predictions; and Section V concludes this paper.

## II. BACKGROUND

The work presented in this paper deals with the QMCPack code, its extension to collect fine-grained statistics, and the integration of the ADIOS library into the code. It also deals with the Skel tool to collect I/O times of QMCPack using ADIOS without the need for running entire simulations. In this section we give a brief overview of these three key software components, (i.e., QMCPack, ADIOS, and Skel).

### A. The QMCPack Code

The QMCPack project uses quantum Monte Carlo methods to find solutions to the many-body Schrodinger's equation by stochastic samplings [1]. Quantum Monte Carlo is one of the most accurate methods and has been applied to a wide range of problems from molecules to condensed matter systems. In theory, QMCPack seems to be the perfect exascale application, but as our work shows, the performance can be hindered by its I/O when executed on a large number of nodes.

QMCPacks algorithm works through several phases. First, it randomly generates many trial solutions called walkers. Walkers are then evolved over many steps using a method called Variational Monte Carlo (VMC), which gives rough solutions to the system of particles then refined using many iterations of Diffusion Monte Carlo (DMC). VMC and DMC are subdivided into blocks, and each block is divided into steps. Each step corresponds to one modification of the walker's data and one communication event between processes to compute global variables, such as the trial energy and the current number of walkers. Walkers with low energy levels may be duplicated, and walkers with high energy levels may be culled during the modification period of a step. The number of walkers is rebalanced among processes by communicating walker data between processes at the end of each step. Since the algorithm is loosely coupled and highly parallel, QMCPack with its CUDA and OpenMP versions can theoretically take advantage of parallelism at many levels on hybrid petascale computers, such as Titan. At the same time, the I/O cost can sum up as the number of walkers increases to the point that the time for the walkers' I/O constitutes a large part of the wall-clock time.

### B. The ADIOS Framework

ADIOS is an I/O framework currently being developed at ORNL. It consists of a suite of I/O methods, an easy to use read-and-write API, a set of utilities, and metadata stored in an external XML file [2]. The XML file is parsed on ADIOS initialization; its metadata contains a description of the data generated and information on how the data should be written out to disk. Different I/O methods can be specified in the XML file and selected at runtime, including POSIX, MPI, and MPI_AGGREGATE. ADIOS POSIX is the simplest of the ADIOS methods; it uses the standard POSIX file API. Each process writes to one file, and an extra metadata file is created to reference the data output. POSIX obtains high performance when using few processes since it has low overhead, but for large process counts the metadata server of distributed file systems such as Lustre can become a bottleneck. ADIOS MPI_AGGREGATE is a hybrid method that first aggregates data to a small subset of processes and then uses MPI-I/O to write to disk. By accumulating data, ADIOS MPI_AGGREGATE keeps the load on the Lustre metadata server low and, therefore, continues to perform even with very large numbers of processes. By simply changing a line in the XML file, the behavior of an application can be significantly altered and optimized for the architecture of the file system. The simple API allows for minimal changes to the existing code base while the metadata enables I/O method switching without recompiling the code. These ADIOS features simplify our performance profiling and predictions of QMCPack simulations with a diverse set of I/O methods.

### C. The Skel Tool

Skel is a tool for building the skeleton of an application's I/O by decoupling the I/O component of a complex code from its computation and communication components and by generating the I/O skeleton for benchmarking [3]. Skel takes an ADIOS XML metadata descriptor and a set of parameters and generates C or Fortran source code with the appropriate ADIOS API calls, data generation, and timers. In a second phase, Skel generates all the supporting files needed to benchmark multiple ADIOS I/O methods in one pass. The execution of the benchmarks on real platforms is faster because Skel does not include computation and communication but rather I/O; users can easily and effectively collect a large set of I/O performance information. The original implementation of Skel had a major limitation that reduces its applicability: it only provides a method to specify fixed I/O size per "time step." We address this limitation below and use the extended Skel for our predictions. Note that Skel was validated in previous work [4] and, thus, the validation is not part of the contribution of this paper.

## III. EXTENDING QMCPACK AND SKEL CODES

We took a version of the QMCPack code that outputs only a few statistics and redefined the code's I/O algorithms to output more extensive scientific traces and perform efficient checkpointing. Here, we describe our algorithms and discuss how we integrated them into QMCPack with the HDF5 and ADIOS libraries. We also describe how we integrated the I/O model of QMCPack into Skel for measuring I/O times of QMCPack using ADIOS on extreme-scale computers.

### A. *Collecting Rich Scientific Traces*

In the original version of QMCPack, called QMCPack STATS, each process generates a reduced number of statistics about the walkers' energetics and writes them to disk every $n$ steps, where the number of steps is defined by the user. More specifically, the processes perform an MPI reduce to calculate a system-wide average of eleven energetics across all QMCPack walkers. These average values are written to disk in a single location supported by the HDF5 data and programming model [5]. This procedure creates very little data, incurs almost no I/O overhead, and allows for high scalability. At the same time, the minimal data is often not sufficient to answer scientists' questions with the desired level of detail.

We designed and implemented a more interesting version of QMCPack, called QMCPack TRACES, that increases the granularity of the calculated energetics statistics from an average value taken across the entire simulation to exact values at a per-walker and per-particle level. The increased number of statistics collected includes the positions of the particles, which are necessary to study the density of the particles within the walkers. These modifications required major changes to the QMCPack I/O algorithms to support the additional statistics collected and the finer granularity. To do so, we extended the code with an application level buffer for the collected statistics. Each process saves the contents of its buffer to disk every $n$ steps, where $n$ is provided by the user and can range from one to hundreds of steps. The buffer helped performance in several ways. First, it allowed for larger but less frequent writes, thus, incurring fewer penalties associated with frequent access to the Lustre meta-data server. Second, it aggregated the variable sized writes across multiple time steps, resulting in a more balanced load across processes and fewer stragglers. Our first implementation of QMCPack TRACES used the HDF5 data and programming model.

To integrate ADIOS into the STATS and TRACES versions of QMCPack, we substituted HDF function calls with ADIOS function calls and created the ADIOS XML metafiles, which define how data is stored and written to disk by ADIOS. The analysis of QMCPack codes using ADIOS will be presented in Section IV.

### B. *Performing Effective Checkpointing*

In both QMCPack STATS and QMCPack TRACES codes using HDF5, the checkpoint write was done by having each process communicate its particle positions to a master process using a *MPI_Gather()* call. The data was then written out to a single disk using HDF5. Gathering the data in this way had two primary limitations. First, the node hosting the master process required enough memory to hold all the walkers from all the nodes. Second, as the number of processes increased, the data transfer times increased, eventually impacting the scalability of the simulations on extreme scale machines. To address both issues, we modified the QMCPack code to perform a virtual write-out per process through the ADIOS API. Once the data was handled by ADIOS, we used one of the several ADIOS write methods to efficiently write data to storage, including MPI_AGGREGATE.

QMCPack's original checkpoint-restart implementation also used a master-worker model, where the master read in all the data and then redistributed to each worker. While restart performance is much less critical than write performance, restarting is not feasible if the walkers' data exceeds the memory on the node hosting the master process. Moreover, in the original checkpoint-restart, QMCPack was able to restart from many different starting files, where each file could have been written by a different QMCPack simulation with a different number of processes. This flexibility complicated the ADIOS restart implementation since ADIOS stores data on a per-process basis called a block and QMCPack writes a variable amount of data per process. We took into account these challenges and redefined the I/O algorithms in QMCPack so that, when restarting, all processes can now query the ADIOS metadata file for the size of each of the blocks, calculate which blocks they needed to read in, and identify relevant portions of those blocks. Moreover, each process keeps track of how many walkers the other processes had read in, allowing QMCPack to restart without using any MPI communication while maintaining a balanced distribution of walkers. We integrated the new checkpoint algorithms in QMCPack and used the ADIOS API to select effective I/O methods in Section IV.

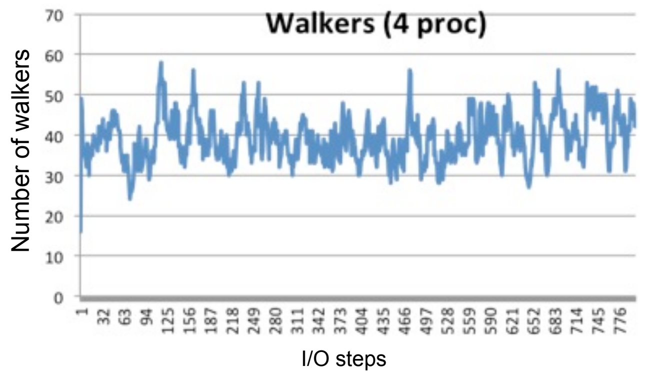
### C. *Emulating QMCPack's I/O*

To predict the impact of I/O on performance for the QMCPack code at a large scale in a short turnaround time, we used Skel. The original Skel tool supported a fixed-size I/O model that is not realistic for many scientific applications, QMCPack included. As described in Section II, QMCPack processes have a variable number of walkers. As a QMCPack simulation evolves, walkers are generated or terminated based on their energy values, making the I/O of each process variable in size. We modeled the variable number of walkers through the behavior of a spring as the number of walkers fluctuated around a target value. We approximated the dampened spring equation using the Midpoint method and then made small random changes to the spring's position. We empirically validated the typical fluctuations of the number of walkers per process against the values generated by the model by tracking their creation and termination and found the patterns to be similar, as shown in Figure 2. The inherent randomness of Monte Carlo simulations precludes an exact match; the approximation is adequate for the work in this paper.

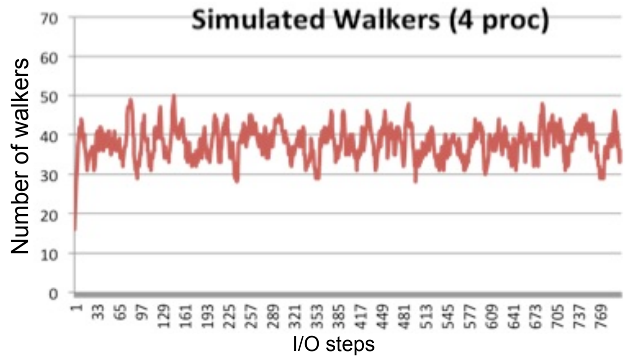
We extended Skel to integrate our spring-based model of the variable QMCPack I/O through inline C and Fortran functions or the inclusion of code from files. This was accomplished by adding additional tags to Skel's XML description of the I/O. When these tags are encountered, Skel uses the user-specified function to generate the variable I/O across processes.

## IV. PERFORMANCE ANALYSIS AND PREDICTION

The I/O performance of the QMCPack codes with the original HDF5 library and ADIOS are measured and analyzed



(a) Empirical QMCPack walkers' numbers



(b) Spring-based simulated numbers

Fig. 2: Comparison of empirical QMCPack walkers' numbers versus spring-based simulated numbers.

below. The predictions of the I/O impact on QMCPack simulations when executed on extreme-scale computers end this section. We conducted our tests on Titan, a Cray XK7 machine with 18,688 compute nodes connected to Spider, a Lustre filesystem with 10.7PB of space and a bandwidth of 240GB/s. We considered three molecular systems different in size: graphite 3x3x1 with 36 carbons and 144 electrons (showed in Figure 3), graphite 4x4x1 with 64 carbons and 256 electrons, and graphite 4x4x2 with 128 carbons and 512 electrons.

## A. *Performance Analysis*

To analyze the I/O performance of the STATS and TRACES codes, we compared the weak-scaling I/O performance of the original version of the QMCPack code using HDF5 with our modified versions of the same code utilizing the ADIOS methods POSIX and MPI_AGGREGATE. For our tests we used 2, 4, and 8 aggregators for ADIOS MPI_AGGREGATE. Each run had two MPI processes per node with eight OpenMP threads per MPI process. We ran tests with 512, 1024, 2048, and 4096 nodes, or in other words 1024, 2048, 4096, and 8192 processes.

The STATS versions of QMCPack with ADIOS and HDF5 exhibit similar performance behaviors as QMCPack with no I/O because of the small amount of data written per iteration. We observed I/O overheads consistently below 1% for both ADIOS and HDF5 versions of the code. Because this comparison does not add any new I/O insight, it is not further

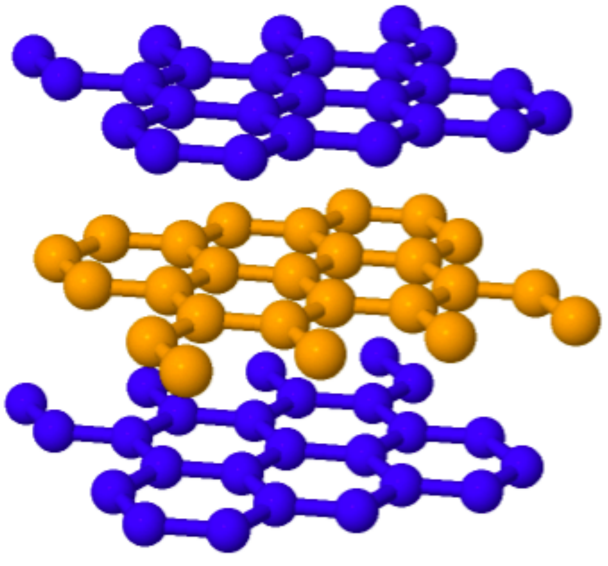
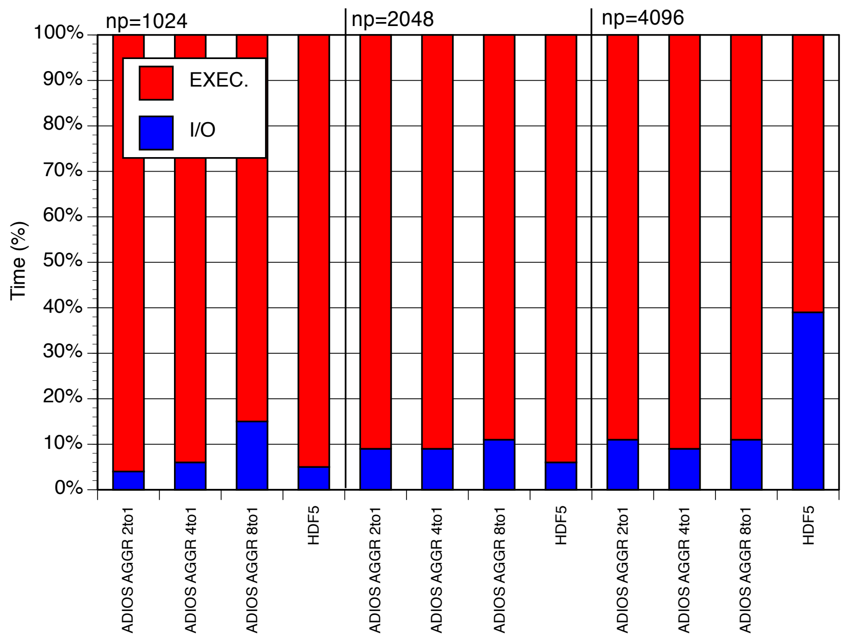


Fig. 3: Graphite 3x3x1 with 36 carbons and 144 electrons.

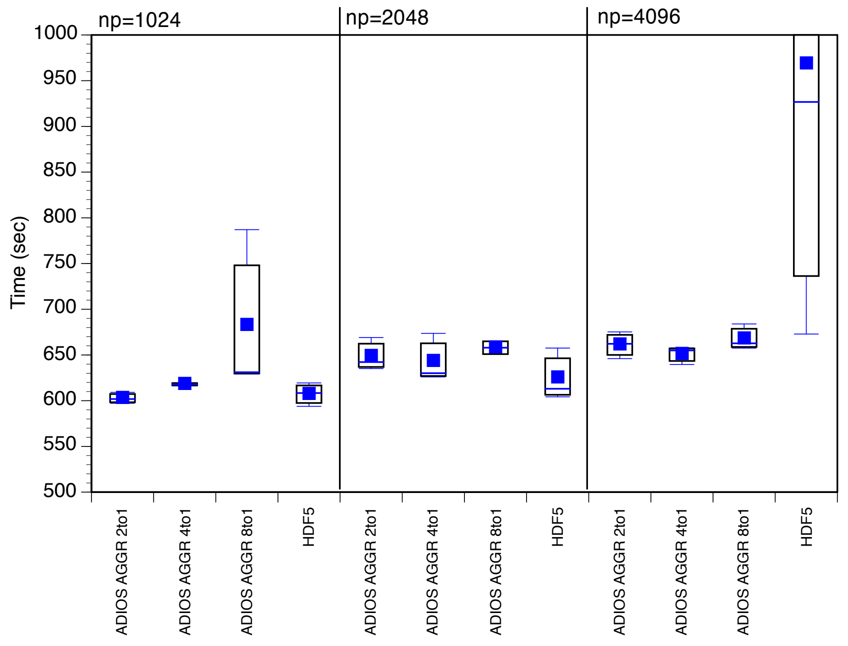
discussed in this paper.

On the other hand, the version of QMCPack TRACES writes larger amounts of data than QMCPack STATS, more frequently. Graphite 4x4x2 writes between 12GB and 50GB every 60s; graphite 4x4x1 writes between 3GB and 24GB every 12s; and graphite 3x3x1 writes between 1.5GB and 13GB every 4s depending on the number of nodes used on Titan. These frequencies and data sizes reveal an important scaling trend. The data sizes scale linearly with the number of particles in the systems, but the write frequencies scale polynomially. This is due to the $O(n^2)$ runtime of the QMCPack's algorithm, where $n$ is the number of particles. We observed that the larger the simulated system, the lower the I/O overhead. We also observed that for small molecular systems, such as graphite 4x4x1 and 3x3x1, smaller, more frequent writes result in inefficient scaling for both ADIOS and HDF5. Graphite 4x4x2, simulations with ADIOS MPI_AGGREGATE and HDF5 exhibit similar performance when executed on a small number of processes and reach our goal of less than 10% I/O overhead despite the larger I/O. For a larger number of processes, we observed that ADIOS exhibits better performance than HDF5 when the user defines an optimal number of cores to aggregate and schedule the data to be written. No substantial motivation was found for the increased variability of HDF5 with 4096 processes, and the further investigation of the problem is work in progress.

These trends for the larger graphite 4x4x2 are outlined in Figures 4(a) and 4(b). Figure 4(a) shows the average percentage of time spent in execution (i.e., for computation and communication) versus the time spent for I/O. The values are obtained over a set of 6 QMCPack runs, each performing 100 DMC steps and printing in output the traces every 10 steps. Figure 4(b) shows the variability of the times due to I/O across the sampled set in a box plot. The top, middle and bottom line of each box corresponds to the 75th percentile (top), 50th percentile (middle) and 25th percentile (bottom).



(a)



(b)

Fig. 4: Percentage of time spent on the execution of QMCPack TRACES versus I/O (a) and time variability due to I/O across samples (b).

TABLE I: Size of data written to disk per checkpoint step for each of the three graphite systems.

| Graphite | 3x3x1 | 4x4x1 | 4x4x2 |
|---|---|---|---|
| 2048 nodes | 921.6MB | 1636.8MB | 3257.6MB |
| 4096 nodes | 1835.2MB | 3238.4MB | 6532.8MB |

TABLE II: Average checkpoint times for QMCPack with ADIOS on different platform configurations.

| | Petascale (Titan, 16K nodes) | Extreme-scale (64K nodes) |
|---|---|---|
| Time (sec) | 1.7sec | 5sec |

The small square indicates the arithmetic mean of the sampled times. Note that because ADIOS POSIX performs very well with a smaller number of nodes but fails to scale past 512 nodes, the figure presents only results of HDF5 runs and the ADIOS MPI_AGGREGATE method using different numbers of aggregates.

The checkpoint version of QMCPack with ADIOS clearly outperforms the HDF5 version because of the different approach our version of the code applies to write the checkpoint data to disk, as described in Section III-B. Preliminary results on smaller systems, such as water, pointed out the performance deficiencies of the HDF5 code as shown in Figure 1, making checkpointing for graphite systems unfeasible on extreme-scale computers. Thus, we did not further use the HDF5 version in the analysis of the checkpoint performance. Our tests were performed with the CUDA version of QMCPack on 2,048 and 4,096 nodes of Titan. Each run used one single process per node and is executed on the nodes GPU. Table I shows the amount of data written by QMCPack per checkpoint with the three different molecular systems (i.e., graphite 3x3x1, 4x4x1, 4x4x2). When checkpointing the particle positions of a small graphite 3x3x1 with our QMCPack code using ADIOS, POSIX performs better on a small number of nodes, but as the number of nodes increases the MPI_AGGREGATE method does increasingly well. For the medium graphite 4x4x1, while MPI_AGGREGATE continues to do well, we observed high variability for the POSIX runs. For the large graphite 4x4x2, MPI_AGGREGATE continues to perform better than POSIX. The checkpoint times for the large graphite 4x4x2 are presented in Figure 5(a). The figure shows the variability of the times per checkpoint in a box plot. Overall, all runs exhibit time variability for the various I/O methods. MPI_AGGREGATE and large molecular systems performance results are the most consistent across runs and, thus, used in the next section for estimating the checkpoint impact on QMCPack simulations on extreme-scale computers. We believe the variability is due to two factors: the loads on the metadata server and on the Lustre Object Storage Targets (OSTs) as the runs were done on shared clusters where the OSTs may have been highly loaded.

## B. Performance Predictions

To estimate the performance impact of I/O associated with checkpointing of QMCPack simulations when preventive and proactive tactics for checkpointing [6] are used, we need to refer to credible platform configurations for the predictions. According to Cappello and co-workers [7], a petascale machine like Titan has an expected mean time to failure (MTTF) ranging from 24 hours to 6 hours. Again based on [7], to minimize the loss in computation due to a simulations crash, a preventive tactic should write the checkpoint data with a frequency of 30 minutes, while a proactive tactic is expected to checkpoint every 10 to 5 seconds. For extreme-scale configurations, Cappello and co-workers expect a MTTF of 2 to 1 hours in an optimistic scenario with a write frequency of every 2.5 minutes for a preventive tactic and of 5 to 1 sec for a tactic checkpointing. In a pessimistic scenario, a MTTF can be experienced every 30 minutes and, thus, a preventive tactic will write every 10 minutes to disk; a proactive tactic for checkpointing should write every 5 to 1 sec. As proven in Figure 1 where similar frequencies were used, these frequencies are unthinkable on Titan for QMCPack simulations using HDF5, much less for extreme-scale platforms. The question we want to address here is whether ADIOS can enable such checkpoint frequencies and at what cost. To answer the question we refer to the numbers in [7] for our estimations.

Our goal is to estimate the checkpoint time in QMCPack with ADIOS at a large scale (i.e., on all the nodes of Titan as well as on an extreme-scale configuration). To this end, we applied linear extrapolation techniques onto a dataset of observed checkpoint times to estimate times beyond the original dataset samples. We generated the dataset from QMCPack simulations using ADIOS MPI_AGGREGATE as presented in Figure 5(a) and from Skel runs presented in Figure 5(b). The QMCPack times were generated over a range of 512 - 2,048 nodes and consisted of 220 write samples. Note that we considered the ADIOS MPI_AGGREGATE method for our sampling because it is stable and generally performs better. On the other hand, Skel times were generated over a range of 4,096 - 16,384 nodes and consisted of 660 write samples. The linear interpolation of the sampled times gave the function: $time_{chkpnt} = 0.71054612 + n * 0.06765666$, where $n$ is the number of processes considered. Table II summarizes the average checkpoint times for QMCPack with ADIOS on 16,384 processes on Titan as well as on a platform that is one order of magnitude larger than Titan in terms of number of nodes.

With the values in Table I, we computed the percentage of time spent on execution (i.e., computation and communication) versus I/O for checkpointing over a period of 24h on Titan for QMCPack on 16,384 nodes. The frequency of the checkpointing is given by the length of the day in seconds (86400 seconds) over the frequencies for the preventive tactic (i.e., every 30 minutes) and for the proactive tactic (i.e., every 10 seconds). The time spent for checkpointing over the 24 hours is given by the checkpoint frequency multiplied by the checkpoint times found in Table I. Similarly, we can compute the same times for the optimistic and pessimistic extreme-scale configurations where we assume the size of the platform almost one order of magnitude larger than the petascale configuration of Titan. More specifically, we assume the extreme scale platform comprising of 65,536 nodes. Figure 6 shows the expected average times in percentage for execution and I/O for the three configurations. We observed how a preventive

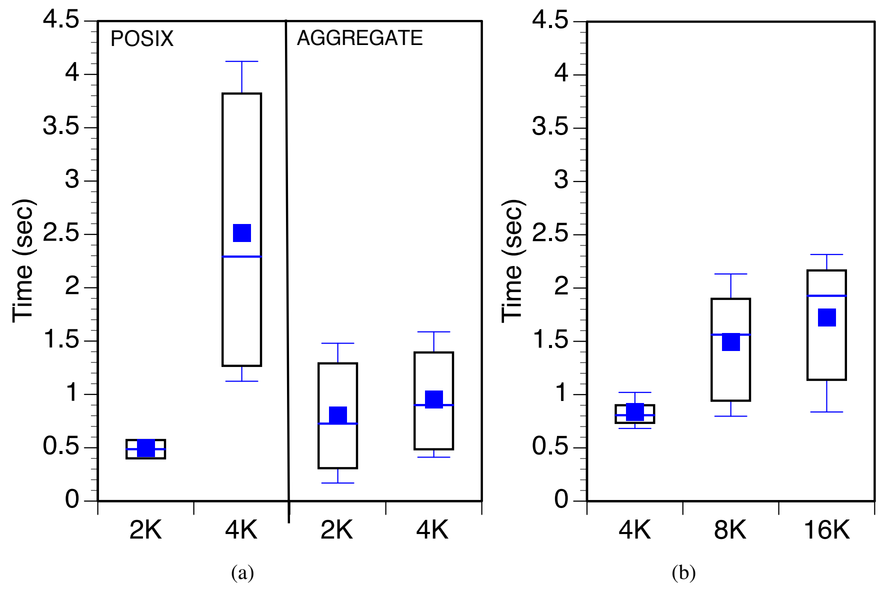


Fig. 5: Checkpoint times for the larger graphite 4x4x2 with QMCPack using ADIOS (a) and Skel (b).

tactic is expected to have a negligible impact (less than 5%) on the I/O cost of QMCPack checkpointing for both an existing petascale platform with 16K nodes, such as Titan, and for a theoretical extreme-scale platform with 64K nodes. This is not the case for a proactive tactic for which a QMCPack simulation would spend almost 18% of the time checkpointing on Titan and 100% of the time on the extreme-scale platform.

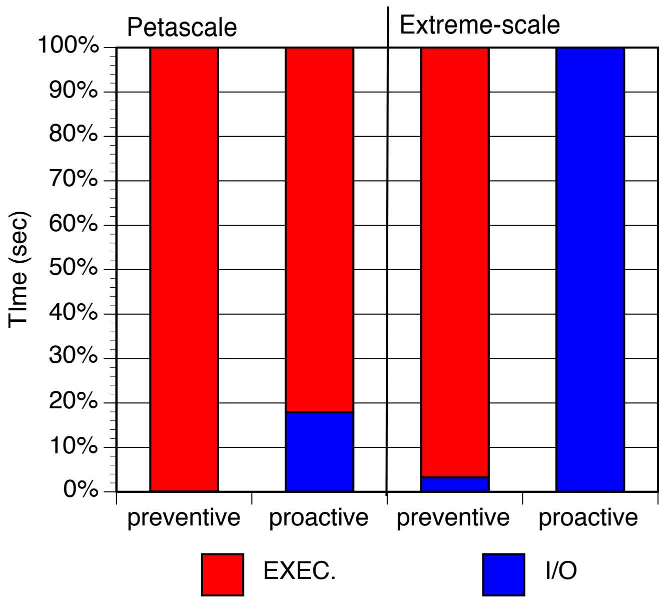


Fig. 6: Predicted percentage of time spent for execution and I/O on a petascale platform with 16K nodes, such as Titan, and on a theoretical extreme-scale platform with 64K nodes.

## *C. Discussion*

In the comparisons of QMCPack performance with ADIOS and HDF5, there is no doubt that the ADIOS version of the checkpointing QMCPack code enables effective checkpointing while the HDF5 version showed costs that were too high.

While in some of the TRACES tests presented above ADIOS and HDF5 exhibit similar performance, as shown in Section IV-A, there are three main reasons for relying on the first rather than the second I/O model. First, ADIOS allows for easily adjustable I/O-methods and parameters. In other words, users can tweak I/O parameters and methods without having to recompile their codes. This feature is particularly useful when optimizing codes on a new system, (e.g., when moving simulations from Titan to Blue Gene/Q). ADIOS only requires changing some parameters in the ADIOS XML file to optimize the I/Os code for a specific system, (e.g., variable number of aggregators for the MPI_AGGREGATE method) and no recompilation is required. Optimal performance with HDF5, on the other hand, may require tweaking of the source code.

Second, ADIOS improves file organization and access for analysis and visualization. By creating a single metadata file, ADIOS abstracts away management of the numerous files created by the QMCPack simulations. The users open the single metadata file and execute the read data just as they would if the data were in a single file, leaving the responsibility to handle the heavy work to ADIOS. In contrast, with HDF5, users have to handle the opening of every single HDF5 file and then coordinate the reads potentially across multiple nodes, making the data access harder due to the manual MPI code. Moreover, ADIOS automatically generates and saves additional information about the data in the metadata file, including minimum, maximum, and average values of I/O variables. This information is available to the users for free during the analysis. The version of QMCPack using ADIOS is now one change away from using staging in which QMCPack traces can be sent to a set of nodes in the system where different applications can analyze and visualize the science generated with the simulations. This can be accomplished by changing the parameters in the ADIOS XML file without any code changes. In contrast to ADIOS, HDF5 would require the QMCPack developers to duplicate the HDF5 code and customize it for any staging method they choose to use.

Third, QMCPack using ADIOS can potentially benefit from the new ADIOS I/O methods that are being released. For example, the work of Tian and co-workers [8] introduces a time coalescing method in ADIOS that allows for a finer grained write in output with no write performance losses. The method allows users to write data of multiple time steps in output all at a single time, preventing the generation of big buffers for the accumulation of multiple time steps. The method also allows ADIOS better indexing of the data, thus improving read performance, and adds a transformation layer to chunk the variables across the time dimension. This can greatly improve read performance of QMCPack values across its time steps. This and other in-progress ADIOS transformations allow for performance enhancements in QMCPack simulations: all QMCPack developers have to do is add a new parameter to their ADIOS XML file before starting their simulations.

## V. CONCLUSION

In this paper we described our work to achieve two main goals: extending the QMCPack code to efficiently support fine-grained writing of scientific data to disk and estimating the impact of I/O on checkpointing for the QMCPack code using ADIOS on petascale platforms and beyond.

We achieved the first goal by redesigning the I/O algorithms of QMCPack to write positions and energies of each single particle rather than average energies of the whole molecule. We assessed the success of our work by analyzing the performance of QMCPack when using either HDF5 or ADIOS. Both HDF5 and ADIOS kept the I/O overhead below 10% on Titan with simulations with up to 4096 processes. As the number of processes increased, we observed an initial slowdown due to I/O that is larger with HDF5. A further study of the slowdown is work in progress. Although in our tests we did not observe any significant performance difference when using ADIOS or HDF5, we believe that ADIOS may be preferable beyond petascale because it comes with the additional features discussed in the paper.

We achieved the second goal by using linear extrapolation techniques to model the checkpoint times and by predicting the percentage of time a simulation of 24 hours would spend checkpointing when using either a preventive or proactive tactic for checkpointing in both optimistic and pessimistic scenarios. We showed how ADIOS enables frequent checkpointing with negligible overheads (below 5%) when using preventive tactics for checkpoint on petascale and beyond.

## ACKNOWLEDGMENTS

S. Herbein, M. Matheny, and M. Wezowicz equally contributed to this work. This work is supported in part by the NSF grants: CNS 1318417 and CNS 1217812. It was also partially supported by the projects Predictive Theory and Modeling for Materials and Chemical Science by the Department of Energy (DOE), Basic Energy Science (BES), the Laboratory Directed Research and Development Program of ORNL, and the DOEs SULI Program. The authors also want to thank Dr. Norbert Podhorszki for his advice on using ADIOS.

## REFERENCES

[1] J. Kim and K.P. Esler and J. McMinis and M. A. Morales and B.K. Clark and L. Shulenburger and D.M. Ceperley. *Hybrid Algorithms in Quantum Monte Carlo.* Journal of Physics: Conference Series, 402(1): 012008, 2012.

[2] J.F. Lofstead and S. Klasky and K. Schwan and N. Podhorszki and Chen Jin. *Flexible I/O and Integration for Scientific Codes through the Adaptable I/O System (ADIOS).* CLADE 2008, 15-24, 2008.

[3] J. Logan and S. Klasky and H. Abbasi and Q. Liu and G. Ostrouchov and M. Parashar and N. Podhorszki and Y. Tian and M. Wolf. *Understanding I/O Performance Using I/O Skeletal Applications.* Euro-Par 2012, 77-88, 2012.

[4] J. Logan and S. Klasky and J.F. Lofstead and H. Abbasi and S. Ethier and R.W. Grout and S.-H. Ku and Q. Liu and X. Ma and M. Parashar and N. Podhorszki and K. Schwan and M. Wolf. *Skel: Generative Software for Producing Skeletal I/O Applications.* e-Science Workshops 2011, 191-198, 2011.

[5] The HDF Group. Hierarchical data format version 5, 2000-2010. http://www.hdfgroup.org/HDF5.

[6] F. Cappello and H. Casanova and Y. Robert. *Preventive Migration vs. Preventive Checkpointing for Extreme Scale Supercomputers.* Parallel Processing Letters, 21(2): 111-132, 2011.

[7] M. Slim Bouguerra and A. Gainaru and L. Bautista Gomez and F. Cappello and S. Matsuoka and N. Maruyam. *Improving the Computing Efficiency of HPC Systems Using a Combination of Proactive and Preventive Checkpointing.* IPDPS 2013, 501-512, 2013.

[8] Y. Tian and S. Klasky and Y. Weikuan and H. Abbasi and W. Bin Wang and N. Podhorszki and R. Grout and M. Wolf. *SMART-I/O: SysteM-AwaRe Two-Level Data Organization for Efficient Scientific Analytics.* MASCOTS 2012, 181-188, 2012.