SAND2018-0299R

EXASCALE COMPUTING PROJECT

ECP-U-2017-XXX
SAND 2018-XXXX

SPARC: Demonstrate burst-buffer-based checkpoint/restart on ATS-1

WBS 2.3.4.04 STDV04-SNL ATDM Data and Visualization Projects, Milestone ST-MW-05-1300

Authors: Ron Oldfield, Craig Ulmer, Patrick Widener, Lee Ward

Author Affiliation: Sandia National Laboratories

January, 2018

U.S. DEPARTMENT OF ENERGY | Office of Science

National Nuclear Security Administration

ECP-U-2017-XXX

ECP Milestone Report
SPARC: Demonstrate burst-buffer-based checkpoint/restart on ATS-1
WBS 2.3.4.04, Milestone ST-MW-05-1300

Office of Advanced Scientific Computing Research
Office of Science
US Department of Energy

Office of Advanced Simulation and Computing
National Nuclear Security Administration
US Department of Energy

January, 2018

ECP Milestone Report
SPARC: Demonstrate burst-buffer-based checkpoint/restart on ATS-1

WBS 2.3.4.04, Milestone ST-MW-05-1300

APPROVALS

Submitted by:

Ron A. Oldfield, Manager, Scalable Analysis and Visualization Sandia National Laboratories	Date

Concurrence:

Author, Title Affiliation	Date

Approval:

Author, Title Affiliation	Date

REVISION LOG

Version	Creation Date	Description	Approval Date
1.0	01-04-2018	Original	

CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Recent high-performance computing (HPC) platforms such as the Trinity Advanced Technology System (ATS-1) feature burst buffer resources that can have a dramatic impact on an application's I/O performance. While these non-volatile memory (NVM) resources provide a new tier in the storage hierarchy, developers must find the right way to incorporate the technology into their applications in order to reap the benefits. Similar to other laboratories, Sandia is actively investigating ways in which these resources can be incorporated into our existing libraries and workflows without burdening our application developers with excessive, platform-specific details.

This FY18Q1 milestone summaries our progress in adapting the Sandia Parallel Aerodynamics and Reentry Code (SPARC) in Sandia's ATDM program to leverage Trinity's burst buffers for checkpoint/restart operations. We investigated four different approaches with varying tradeoffs in this work: (1) simply updating job script to use stage-in/stage out burst buffer directives, (2) modifying SPARC to use LANL's hierarchical I/O (HIO) library to store/retrieve checkpoints, (3) updating Sandia's IOSS library to incorporate the burst buffer in all meshing I/O operations, and (4) modifying SPARC to use our Kelpie distributed memory library to store/retrieve checkpoints.

Team members were successful in generating initial implementation for all four approaches, but were unable to obtain performance numbers in time for this report (reasons: initial problem sizes were not large enough to stress I/O, and SPARC refactor will require changes to our code). When we presented our work to the SPARC team, they expressed the most interest in the second and third approaches. The HIO work was favored because it is lightweight, unobtrusive, and should be portable to ATS-2. The IOSS work is seen as a long-term solution, and is favored because all I/O work (including checkpoints) can be deferred to a single library.

1. INTRODUCTION

This report documents completion of milestone "SPARC: Demonstrate burst-buffer-based checkpoint/restart on ATS-1", listed as STDA04-10 with the milestone ID ST-MW-05-1300 in the ECP Jira site. One of the primary objectives of Sandia's ATDM Data and Visualization project is to provide application support for the ATDM applications of interest to Sandia's ASC mission. For this milestone, the objective is to demonstate a checkpoint/restart capability for the Sandia Parallel Aerodynamics and Reentry Code (SPARC) that leverage the non-volatile-memory (NVM) burst-buffers available on the ATS-1 system at LANL.

Demonstrating a performant checkpoint/restart capability using burst-buffers addresses two important issues for our ATDM and ECP program. First, application resilience is a requirement to ensure progress for our mission-critical codes on our large-scale systems. Application-directed checkpoint/restart is the most common approach used by our codes and providing that capability is an important step toward developing a production-capable code. Second, burst buffers are a fairly new technology on our large-scale systems. Developing the software abstractions that enable applications to leverage these burst buffers in a portable way allows a more rapid integration of this technology into other ATDM and ECP codes, and it provides a way to evaluate and compare performance of these new technologies.

2. MILESTONE OVERVIEW

2.1 DESCRIPTION

This milestone demonstrates a checkpoint/restart capability for the Sandia Parallel Aerodynamics and Reentry Code (SPARC) that leverage the NVRAM burst-buffers available on the ATS-1 system. This work will builds on the checkpoint/restart prototype developed in FY17 into a stable capability that SPARC users can leverage in production jobs. The checkpoint/restart code will (1) interact with the burst buffers to decrease the overhead of checkpointing and (2) coordinate data exchanges between the burst buffers and the parallel file system.

2.2 EXECUTION PLAN

The execution plan involves four different approaches, in increasing complexity:

1. Stage-in/Stage-out: This approach uses simple scripts to direct existing I/O to burst buffers. This should be a relatively simple solution, but will likely not be portable across HPC systems.

2. Use the HIO library from LANL: The HIO library is a bust-buffer library developed by LANL staff. It is intended to be portable across HPC systems, but it has an overly simplistic view of the memory and storage hierarchy (e.g., it has no memory tier). This would also be a SPARC-specific solution. In other words, the API we generate for SPARC would have little relevance to other codes that need a checkpoint/restart capability.

3. IOSS+HDF5: The third approach is to use Sandia's commonly-used I/O library IOSS, which uses an HDF5 backend. This requires an HDF capability to use the Trinity burst buffers. The advantage of this approach is that it provides a capability that would be immediately available to other applications that use IOSS, including a good portion of our ASC integrated codes.

4. Kelpie: The final approach is to develop code that would checkpoint to Sandia's research-grade intermediate data-management layer, Kelpie. The long-term advantage of this approach is that we would be able to offload or direct I/O and analysis to external components (e.g., visualization tools). This approach goes beyond what is requested by the application team, but helps us sustain some of our more research-driven technologies.

2.3 COMPLETION CRITERIA

As evidence of completion, we will integrate our code into SPARC code base, provide performance measurements for checkpointing/restarting job on Trinitite, and provide scripts for demonstrating how to launch a job, checkpoint it, and then relaunch the job starting from the checkpoint. To ensure progress and completion, the team will require continued access to SPARC codebase and developers. Early notification of any significant changes to SPARC I/O, and access to burst buffers on Cray systems.

2.4 MILESTONE DEPENDENCIES

2.4.1 Milestone Predecessors

No prior milestone completion required.

3. TECHNICAL WORK SCOPE, APPROACH, RESULTS

3.1 STAGE-IN/STAGE-OUT

The first approach we explored in this work was simply to add DataWarp staging directives to SPARC's SLURM batch job scripts to redirect checkpoint/restart data to the burst buffer. This approach generally provides the easiest way for application developers to retrofit an existing application to work with burst buffers because platform parameters are adjusted in the job script instead of the actual application. It was straightforward to adapt SPARC to work with this approach because the SPARC executable defaults to using the local directory to write/read state data. As such, the jobs script needed to be modified to perform the following:

- **Allocate Burst Buffer Space**: DW directives are used to allocate space in the burst buffer for checkpoint data.
- **Stage In**: DW directives are then used to move any existing checkpoint data from the parallel filesystem to the burst buffer.
- **Change Directory and Execute SPARC**: The compute nodes in the job change directory to the local mount point for the burst buffer and then launch the SPARC executable with a supplied input deck. SPARC defaults to performing I/O in the local directory and therefore use the burst buffer for I/O.
- **Stage Out**: DW directives in the job script are used to migrate SPARC data from the burst buffer to a location in the parallel filesystem when the application completes.

After demonstrating that a static job script could be written to migrate SPARC data through the burst buffer, we constructed a general script that would make it easier for users to adjust the job parameters on the command line. The job scripts are included in APPENDIX A.

Strengths/Weaknesses: The main strengths of this approach are that (1) it is easy to implement and (2) staging time may not be factored in as part of a job's run time. The weaknesses are that (1) DW directives are currently specific to Cray and (2) the burst buffer may fill if data is not staged out until the end of the job.

Current Status: The work for the stage-in/stage-out approach is complete. However, additional performance benchmarking needs to be performed with larger test sizes.

Lessons Learned: While the DW directives are straightforward to use, they make job scripts more complicated and less portable.

3.2 CHECKPOINT USING HIO LIBRARY

The second approach we explored was updating SPARC's checkpoint/restart code to use LANL's Hierarchical I/O library (libhio) to control how checkpoint data migrates between SPARC, the burst buffer, and the parallel filesystem. Libhio provides a basic hierarchical storage interface that currently has device drivers for both DataWarp and POSIX filesystems. Application developers use libhio to write/read a dataset's components for a particular timestep and then rely on the library to migrate data between storage resources. Policy decisions about how libhio should manage data are defined in an external configuration file. As such, an application that has been adapted to use libhio can be adjusted to use different I/O settings without having to rebuild the application or modify an application's input deck.

We modified SPARC's StructuredAeroModelWrapper class to enable it to write/read checkpoint data using libhio. This work involved extracting key items from SPARC's data structures and serializing them into contiguous objects that libhio could store. While SPARC's use of Kokkos data structures simplified the serialization process (due to regular, contiguous data layouts), a sizable amount of effort went into determining the right place to store out objects.

Strengths/Weaknesses: The primary strengths of this approach are that libhio (1) can migrate objects while the application runs, (2) provides some portability between platforms, and (3) configuration files provide an easy way to adjust I/O parameters for a platform. The weaknesses of this approach are that libhio (1) has a steep learning curve, (2) lacks a memory tier, and (3) is difficult to debug.

Current State: A preliminary version of the libhio checkpoint/restart code for SPARC was demonstrated in late December. There have been code changes in SPARC that will require us to update our code before it can be merged into the SPARC repos. Additional performance measurements will need to be run in order to gain a better understanding of scenarios where the burst buffers will be of use in SPARC.

Lessons Learned: Working with two, evolving code bases made this work more time intensive than we originally planned. In libhio's case, stability issues and a lack of clear examples made it difficult to use. In SPARC's case, it was difficult to keep up with the rapidly evolving code base. In retrospect we would have benefited from defining a standard API for I/O work at the beginning that would shelter us from changes elsewhere in the code.

3.3 CHECKPOINT USING IOSS AND HDF5

The third approach we are exploring for improving SPARC checkpoint performance is to adapt Sandia's IOSS library to take advantage of burst buffer capabilities. IOSS is the main library Sandia applications use to interface with mesh datasets. SPARC currently uses IOSS to load its initial mesh and save its results, but there is interest in also using IOSS to manage checkpoint data. While adapting IOSS to leverage burst buffers would have a positive impact on many of Sandia's applications including SPARC, IOSS is a complex software stack composed of other libraries such as CGNS, Exodus, netCDF, and HDF5. Our approach in this work is to make changes at the HDF5 layer that cause the least disturbances to the rest of the stack.

We initially investigated using DataElevator [1] as a mechanism for incorporating the burst buffer into the HDF5 layer. DataElevator is composed of two parts: (1) a modified HDF vol that routes HDF5 I/O to the burst buffer during a simulation and (2) a special-purpose cleanup job that runs after the simulation and uses MPI-IO to migrate data from the burst buffer to the parallel filesystem. While the example programs from HDF5 ran, we found a fundamental incompatiability between DataElevator and the way IOSS jobs write files. Specifically, DataElevator uses rank 0 to write out a metadata file that describes which HDF5 files are written in the simulation. IOSS ranks write their ouput files independently (i.e., one file per rank), and do not share knowledge about how large individual files are. As such, the current implementation of DataElevator did not provide a way to properly track IOSS files, resulting in the stage out only migrating rank 0's data. It is expected that this deficiency could be expected in future versions of DataElevator by improving its metadata management.

Our second approach was to simply use DataWarp primitives in the SLURM job to control writes to the burst buffer and then stage the files out to the parallel filesystem. While the IOSS test applications correctly wrote data to the burst buffer, the job repeatedly crashed during the DataWarp stage out. While we theorize that there may be an issue with how IOSS/HDF5 is closing out files, we do not have a clear understanding as to why this operation does not work properly.

Our third approach in this effort was to construct our own HDF vol to intercept I/O and route it through the burst buffer to the parallel filesystem. The intercept portion of this work prepends the path for the burst buffer mount point to the filename supplied by the user during open or create. The interception also performs a number of checks to guard against erroneous scenarios (e.g., writing to symbolic links). We experimented with three alternatives to staging out data: (1) using the DataWarp job directives in the SLURM job, (2) calling the DataWarp stage out C functions and blocking when close is called, and (3) calling the DataWarp stage out C functions and not blocking when close is called. The first two options have proven to be the most stable. Experiments with the third option appear to be stable and are expected to yield better asynchronous performance.

Strengths/Weaknesses: The primary strength of this approach is that Sandia applications such as SPARC already use IOSS and can benefit from updates that allow the library to leverage burst buffers. The weaknesses of this approach are that (1) additional effort will be required to ensure the IOSS maintains production quality requirements and (2) checkpoints made by IOSS will likely be larger and slower than customized checkpoints.

Current State: The HDF vol provides basic features necessary to use the burst buffer, but will require more work to be used in production by IOSS. Additional modifications will need to be made in SPARC in order to manage checkpoint data with IOSS. This work is ongoing and is expected to complete by Q4.

Lessons Learned: We encountered a mismatch of design assumptions between DataElevator and IOSS that was unexpected and is difficult to resolve. Jumping into the HDF code to diagnose and fix these problems consumed a considerable amount of effort and slowed our progress.

3.4 CHECKPOINT USING KELPIE

The fourth approach we examined was to update SPARC to use SNL's Kelpie software to manage checkpoints. Kelpie is a distributed memory service being developed in ATDM that provides a flexible way to move datasets between memory, burst buffer, and persistent storage resources. In addition to being able to efficiently transfer data between ranks in a single job, Kelpie provides support for exchanging data between the ranks of different jobs running on the same platform. These capabilities will enable platform engineers to develop the new services for ECP's workflow, analytics, and application coupling needs.

Kelpie is currently at a research quality of technical readiness and needed improvements in order to be connected to SPARC. Kelpie received two important improvements this quarter. First, Kelpie was enhanced to provide better asynchronous handling of remote communication operations. This work involved updating the data object transfer protocol to stall a get operation when there is an object cache miss at the the remote side. Second, a new I/O module (IOM) infrastructure was created to allow data objects to be exchanged with storage devices. Our first IOM in this infrastructure enables Kelpie applications to write/read objects to/from POSIX filesystems. This enhancement provides Kelpie users with a straightforward way to add persistence to their applications.

SPARC was updated to have a preliminary interface for writing checkpoint data to Kelpie. This work was similar to the libhio work described in Section 3.2 with the exception that the SPARC data components that were serialized were packed into labeled Kelpie objects. We found this serialization to be more natural because we did not have to manage explicit dataset offsets or C++/C conversions the way we had to with the libhio approach. Data objects written into Kelpie are written by the IOM to a POSIX mount point (either the parallel filesystem or a burst buffer). We verified that SPARC objects are stored correctly in both memory and disk. The next step for this work is to construct a corresponding reader that will enable checkpoint data to be loaded into SPARC.

Strengths/Weaknesses: The strengths of the Kelpie checkpointing work are that it (1) provides a flexible means of checkpointing data to different resources, (2) enables developers to use Kelpie's interfaces to inspect a running application, and (3) serves as a starting point for connecting other applications to SPARC. The primary weaknesses for this approach are that (1) it is overkill for what SPARC users currently require and (2) additional hardening will be required to meet production requirements.

Current State: The current implementation provides a basic way to store checkpoints, but lacks a means of loading the checkpoint data to restart the simulation. The recent refactoring of SPARC components will also require changes to the Kelpie SPARC hook before the work can be commited into the SPARC repository.

Lessons Learned: We found that it was easier to write the SPARC checkpointing functions with Kelpie than it was with libhio because Kelpie employs a higher level of abstraction. While preparing the checkpoint/restart demonstration of the code to the SPARC team we became aware that Kelpie provided a side benefit for monitoring and debugging the state of the running application.

4. RESOURCE REQUIREMENTS

Table 1 Estimated resources consumed

Resource	Estimated Usage
People Effort	9 person-months (6 half-time people for three months)
HPC resources	1000 core-hours on ATS development platforms
Materials and supplies	Standard office equipment: $5K
Miscellaneous	Maintanence of repo/dashboard/webpage: $11K Travel: $9K

5. CONCLUSIONS AND FUTURE WORK

During FY18Q1 we performed an initial investigation into four different ways that SPARC could be adapted to use Trinity's burst buffers for checkpoint/restart. While performance measurements need to be made for larger test sizes, the work was sufficient enough to gain insight into different approaches to updating applications to use the burst buffers. While adding DataWarp directives to jobs does not require code changes, we are hesitant to rely on this approach alone due existing complexities in some of our job scripts and the threat of being tied to a specific vendor. Creating a new, application-specific checkpoint/restart capability took time but was greatly simplified by using a burst buffer aware library such as libhio. Adding burst buffer support to IOSS is a long term solution that will have the broadest impact at Sandia. However, this work will take more investment and testing due to the diverse capabilities of IOSS. Finally, we demonstrated our Kelpie distributed memory library can be used to route data to the burst buffer and has additional capabilities that may be useful in more complex workflows.

This work will continue into the following quarters this year. Our plan is to update the libhio work in order to address refactoring that took place in SPARC and then conduct performance experiments on Trinity to obtain statistics on how well it performs. We will also transition the work to run on the ATS-2 platform when the hardware stabilizes and libhio is ported. The IOSS work will also continue throughout the year in order to stabilize the burst buffer modifications. Finally, the Kelpie portion of this work will largely shift to supporting the ATDM EMPIRE code. This work will create new checkpoint/restart capabilities for EMPIRE and conduct performance experiments.

6. ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the Office of Science and the National Nuclear Security Administration—responsible for the planning and preparation of a capable exascale ecosystem—including software, applications, hardware, advanced system engineering, and early testbed platforms—to support the nation's exascale computing imperative.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Also acknowledge any other assistance provided by other staff, projects, institutions (e.g., HPC resources), programs, etc.

7. REFERENCES

[1] B. Dong, S. Byna, K. Wu, H. Johansen, J. Johnson and N. Keen, "Data Elevator: Low-Contention Data Movement in Hierarchical Storage System," in *IEEE 23rd International Conference on High Performance Computing (HiPC)*, 2016.

APPENDIX A. Example Job Script

The following template was used to launch SPARC jobs while using the burst buffer. An additional bash script was used to take command line arguments and substitute them into the job script template (e.g., @BBCAPACITY@).

```
#!/bin/bash -l

#SBATCH -N @NUMNODES@
#SBATCH -t @TIMELIMIT@
#SBATCH -J blunt_wedge
#SBATCH -C haswell
#DW jobdw type=scratch access_mode=striped capacity=@BBCAPACITY@

#DW stage_in type=directory source=@STAGEINDIR@ destination=$DW_JOB_STRIPED/rundir.bb
#DW stage_out type=directory destination=@STAGEOUTDIR@ source=$DW_JOB_STRIPED

## We will copy the input and run the PFS job here.
PFS_RUN_DIR=rundir.pfs.@SUBMITTIME@
## We will rename the DW stage in destination to this.
BB_RUN_DIR=rundir.bb.@SUBMITTIME@


cd $DW_JOB_STRIPED
mv $DW_JOB_STRIPED/rundir.bb $DW_JOB_STRIPED/$BB_RUN_DIR
cd $DW_JOB_STRIPED/$BB_RUN_DIR

echo 'BB Started at ' `date`

srun --cpu_bind=core --ntasks=@NUMTASKS@ --ntasks-per-node=@TASKSPERNODE@ \
     --cpus-per-task=2 @SPARCEXE@ -i sparc.inp > sparc.out 2>&1

echo 'BB Ended at ' `date`

## Copy the input deck to a per job directory on the PFS.
cp -a @STAGEINDIR@ @STAGEOUTDIR@/$PFS_RUN_DIR

cd @STAGEOUTDIR@/$PFS_RUN_DIR

echo 'PFS Started at ' `date`

srun --cpu_bind=core --ntasks=@NUMTASKS@ --ntasks-per-node=@TASKSPERNODE@ \
     --cpus-per-task=2 @SPARCEXE@ -i sparc.inp > sparc.out 2>&1

echo 'PFS Ended at ' `date`
```