Phoenix: Memory Speed HPC I/O with NVM

Pradeep Fernando Sudarsun Kannan Ada Gavrilovska Karsten Schwan
Georgia Institute of Technology

***Abstract*—Non-volatile memory (NVRAM) provides excellent opportunities to accelerate I/O in exascale machines. However, naive use of NVRAM devices with current software stacks can expose the performance bottlenecks due to limited device bandwidth. We address this by designing Phoenix (PHX), a NVRAM bandwidth-aware persistent object store for HPC applications that increases the aggregate checkpoint bandwidth by the simultaneous use of NVRAM and DRAM devices, leading to $\sim 12\times$ reduction in the checkpoint time and a $\sim 18\%$ improvement in total simulation time.**

I. Introduction

High performance computing applications produce huge amounts of input/output (I/O) data that gets stored local to the compute nodes. State of the art multi-level checkpoint schemes [1] store frequent application checkpoints on node local storage in a bid to recover from transient node failures. Scientific simulations use local storage to persist the analytics outputs destined to downstream analytics workflows. Therefore, it is important to design HPC I/O mechanisms that have minimal impact on application performance and permit fast checkpoints and analytics output onto local persistent storage.

Using non-volatile memory devices can play an important role for future I/O in HPC systems. NVRAM devices such as 3D-XPoint, Memristors and PCM are byte addressable and are expected to provide 100x faster read/write access compared to their closest memory-based storage alternatives such as PCIe-based SSD. Furthermore, the NVRAM technologies are expected to provide higher density (up to 10x than DRAM) and do not require refresh energy (unlike DRAM), which makes them a viable candidate to use as stable storage in HPC I/O.

However, the limited bandwidth of the NVRAM device going to be a bottleneck in the context of persistent HPC I/O. First, the next-generation HPC engines will have an order of magnitude increase in core counts per node (64-128 cores per node) – increasing the amount of I/O generated within a compute node. Next, the gang I/O nature of the HPC applications is likely to split the NVRAM bandwidth ever further, thus severely lowering the per-core NVRAM bandwidth. Finally, NVRAM technologies like PCM are known to have limited read/write bandwidth($4-8\times$ lower than DRAM), write bandwidth in particular. We attribute the limited read/write bandwidth to the underlying data storage physics of the device. In summary *the poor bandwidth scaling of NVRAM may become a major bottleneck during HPC I/O*.

We present Phoenix (PHX) – an NVRAM-bandwidth aware library for checkpoint I/O. PHX achieves efficiency through the use of memory-centric object interfaces and a device stack specialized for NVRAM. The library provides applications with familiar 'malloc'-like interfaces for object creation, but differs from the former in that it allows the application programmer to tag a created object space with properties related to persistence and versioning. PHX deals with the limited NVRAM bandwidth through simultaneous use of NVRAM and local/ peer nodes' DRAM devices, thus increasing the effective data movement bandwidth. PHX's memory-centric object interface and NVRAM-bandwidth-aware design lead to reduction in the time length of I/O operations in the critical path, associated with the slow NVRAM device. To continue guaranteeing adequate reliability and persistence, DRAM-resident object state is replicated across peer nodes' memory, which is accessible through high-bandwidth interconnects. PHX is implemented and evaluated with several representative HPC applications – 3D Gyrokinetic Toroidal Code (GTC), CM1 and S3D.

II. PHX Checkpoints

PHX presents the concept of *persistent objects* to the HPC applications. All types of HPC I/O are modeled as operations on memory objects accessed via memory-based APIs. Applications declare the expected object characteristics (persistence, versioning) during the initial object allocation. At the heart of the Phoenix is an object tracking runtime that provides object versioning, reliability, and persistence. The base APIs of the Phoenix runtime are as follows:

- **init()** - initialize the Phoenix library runtime.
- **create_obj(key, size, unit_size, properties)** - allocate an object from **main memory** and set its property (persistence, versioning, etc) details; return a memory pointer to the caller.
- **checkpoint_commit** - copy the content of main memory resident checkpointable variables into NVRAM
- **destroy_obj(key)** - de-allocate the memory space of a given object key.
- **finalize()** - clean-up library resources

Figure 1 shows a simple template for implementing coordinated checkpoints with PHX APIs. During the memory allocation programmers explicitly specify checkpoint objects via the property, 'CHECKPOINT=true', and by default, 'checkpoint_commit()' operates on all such checkpoint objects and moves their DRAM copies to NVRAM (checkpoint).

```
char key[]="foo";
/*create checkpointable object instance*/
struct properties prop = {.checkpoint = true, ...};
void *ptr=create_obj(key,SIZE,prop);
.
/*do computation using data_ptr*/
.
checkpoint_commit();
```

Figure 1: Using PHX-C/R service to carry out coordinated checkpoints

Aggregate bandwidth checkpoints. The NVRAM bandwidth is likely to become the most bottlenecked resource during I/O data movements. PHX uses DRAM bandwidth to alleviate the limited NVRAM bandwidth during data output. The key idea is to use aggregate device bandwidth of both DRAM and NVRAM during I/O data movement, thus shortening the critical path data movement time. Towards that end, first Phoenix splits the total critical path I/O data into two parts: DRAM-bound data (Ddata) and NVRAM-bound data (Ndata), while taking device bandwidth ratios, application object access patterns and DRAM capacity budget into consideration. During a bulk I/O data movement, PHX moves a copy of Ddata into a DRAM buffer, and in parallel moves Ndata over to NVRAM. However, the DRAM is volatile and hence saved Ddata has to be quickly moved from DRAM to NVRAM and committed – a critical property for achieving the data persistent property for written I/O data. We refer to this operation as `I/O de-staging`, and the DRAM-resident temporary buffers as `staging buffers`.

Staged data reliability. The naive use of the aggregate bandwidth approach falls short of providing the required data reliability guarantees for I/O, because part of the output data (Ddata) remain in DRAM before being de-staged to NVRAM, making it vulnerable to data loss. However, in current supercomputers, the network interconnects offer point to point bandwidths up to 56 Gbps and these numbers expected to be 100-400Gb/s in future HPC machines. The fast interconnect bandwidth and remote direct memory access (RDMA) stacks make the cost of writing/reading remote DRAM memory cheaper than that of local NVRAM memory. Phoenix uses N=2 replication scheme for its DRAM staged data (Ddata). For each of compute node, we assign a buddy node to act as the remote DRAM node. During the aggregate bandwidth copy, we write staged data to both local DRAM and to the buddy node's DRAM (using RDMA). We show that N=2 replication scheme brings down the Ddata loss probability from 0.98960270 to 0.00000003 – a negligible failure probability, for a simulation run of 48 hours, with 2880 checkpoints (checkpointing once in every 60 seconds).

Energy Model. While PHX C/R speeds up the checkpoint times, it incurs additional energy overheads compared to naive NVRAM checkpoints, due to extra data movements (de-staging/ buddy checkpoints). However the fast checkpoints results in total simulation time reduction, thus energy savings. We analyze the trade-off between data movement energy costs and fast checkpoints using a closed form abstract energy model.

We plot the Figure 2 graph using faithful instance values derived from the recent literature [2] on the subject. We consider two interconnect technologies and buddy distances (buddy being single/ double switches away) for energy overhead line plots. We show energy saving plots for multiple DRAM:NVRAM bandwidth ratios. For an example, the resulting plot shows that it is possible to stage up to ∼750MB of PHX checkpoint data (N=2 replication, buddy one switch away and using OmniPath interconnect) without extra overheads when the DRAM:NVRAM bandwidth ratio is 2:1. Thus careful selection of PHX configuration parameters will lead to net energy savings during the simulation run.

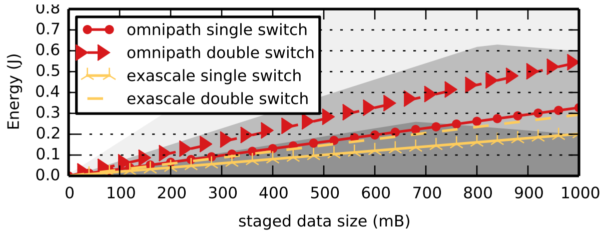

Figure 2: Curved plots (dark, medium and light shaded areas) show the energy savings of PHX C/R over naive NVRAM C/R, under different DRAM : NVRAM bandwidth ratios, 2:1, 4:1 and 8:1 respectively. line plots refer to OmniPath and expected exascale interconnect energy costs, during N=2 replication scheme, while the buddy node distance being single/ two switches away.

III. Evaluation

We evaluate PHX with three real-world HPC applications – GTC, CM1 and S3D (results not presented here due to space constraints) – and observe the C/R performance. The applications consist of algorithms related to different scientific fields, and each application shows different C/R characteristics such as compute to checkpoint data ratios, variable access patterns, etc. We used 4 nodes from the Stampede cluster for our evaluations. Each node consists of dual socket Intel Xeon E5 with 16 cores and 32 GB of main memory, interconnected via 56Gbps Mellanox InfiniBand. We reserve 12 GB out of main memory to NVRAM and emulate the reduced NVRAM bandwidth using software delays.

Experimental Overview. We run each of the HPC benchmarks against three different checkpoint schemes, (i) nvram - a naive method of copying all the data at once during checkpoint (ii) phx - the proposed C/R method which relies on use of aggregate NVRAM + DRAM bandwidth, and finally (iii) phx-ec - corresponding to 'phx' with a speculative pre-copy [3] optimization. We record (i) the checkpoint times of each benchmark application with varying per-core-bandwidth values, (ii) the time spent on de-staging and (iii) the compute iteration time. The results are presented in terms of per-core bandwidth. The metric represent the effective bandwidth seen by one processor core during parallel NVRAM writes. In all experiments we use the N=2 replication scheme for PHX checkpoints and the buddy node is chosen to be one network switch away. We run 12 MPI processes on each node. Each MPI process has a helper thread, but all the helper threads are pinned to a single dedicated core in the node.

GTC [4] checkpoint data constitutes of 2D/3D arrays. It is a data-heavy application and thus naive NVRAM data copy time can be as high as 37% compared to the compute time of the application. Tracking of access times of individual variables suggests that GTC checkpoint variables are pre-copy friendly, that is some of the variables get last accessed early in the compute cycle. Each core writes 235 MB of checkpoint data and we assign a staging buffer size of 125 MB (∼50% of the checkpoint data). The PHX checkpoint scheme cuts the checkpoint time Figure 3 by nearly half and PHX-ec that applies pre-copy optimization over PHX, improves the checkpoint time by as much as 12×.

CM1 [5] represents a highly compute intensive class of applications within out selections of benchmarks. Therefore, naive NVRAM checkpoint time only accounts for ∼7%, compared to compute time. Unlike GTC, CM1 modifies its checkpoint variables up until the checkpoint time. Similarly to the GTC runs, we allocate ∼50% of checkpoint size as the DRAM staging buffer capacity. The numbers in Figure 3 show that PHX-ec fails to deliver any checkpoint time improvement over PHX. However PHX perform 2x better over naive NVRAM checkpoints, proving the effectiveness of the technique, irrespective of the application data access behaviour.

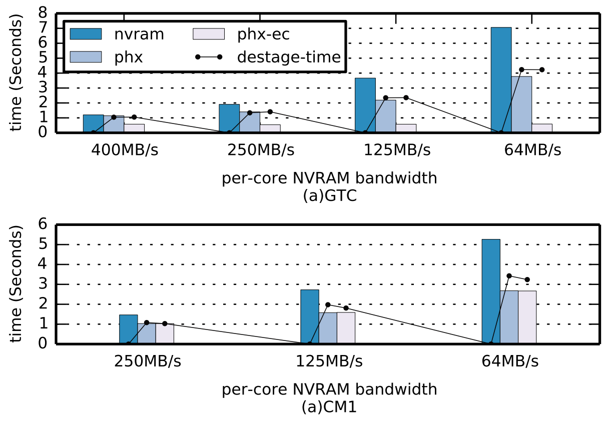

Figure 3: Checkpoint times of workloads, plot against varying per-core NVRAM bandwidths and checkpoint schemes

Conclusion. PHX shortens the NVRAM critical path data movement time and converts DRAM into stable-enough storage for checkpoint data, by a combination of data staging, asynchronous de-staging and data-replication. PHX evaluations on real-world HPC applications and emulated NVRAM hardware shows up to ∼12x speedups in checkpoint times over the naive NVRAM data copy method. The design and evaluation of PHX are presented in detail in [6]

References

[1] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *SC '10*. IEEE, 2010, pp. 1–11.

[2] "Energy Consumption and Performance Design Space Trade-Offs for Optical Data Center Networks," https://arpa-e.energy.gov/sites/default/files/Bergman_Columbia_Data_Center_Workshop_0.pdf.

[3] S. Kannan, A. Gavrilovska, K. Schwan, and D. Milojicic, "Optimizing checkpoints using nvm as virtual memory," in *IPDPS '13*. IEEE, 2013, pp. 29–40.

[4] "Gyrokinetic Toroidal Code," http://phoenix.ps.uci.edu/GTC/.

[5] G. Bryan and J. Fritsch, "A benchmark simulation for moist nonhydrostatic numerical models," *Monthly Weather Review*, vol. 130, 2002.

[6] P. Fernando, S. Kannan, A. Gavrilovska, and K. Schwan, "Phoenix: Memory speed hpc i/o with nvm," in *To be presented in HiPC, 2016*. http://www.cc.gatech.edu/grads/p/pfernand/pubs/hipc2016_pfernando.pdf.