Towards Access Pattern Aware Checkpointing For Kokkos Applications

Nigel Tan
University of Tennessee Knoxville
Knoxville, Tennessee, USA
ntan1@vols.utk.edu

ABSTRACT

The common checkpoint philosophy, checkpoint everything as frequently as possible, is becoming ineffective as we progress towards exascale machines, facing shrinking time between failures. This makes portability and resilience vital for the future of HPC. This poster demonstrates the need and forms the foundation for enhancing checkpointing to take advantage of application properties. Specifically, we show how access pattern aware checkpointing improves performance using incremental checkpoints of sparsely updated data as an example. We also define how the portable checkpointing abstractions in Kokkos Resilience can be modified to support such an enhancement transparently.

CCS CONCEPTS

• **Software and its engineering → Checkpoint / restart**.

KEYWORDS

Graph algorithms, Portability, Resilience, VeloC

ACM Reference Format:
Nigel Tan. 2021. Towards Access Pattern Aware Checkpointing For Kokkos Applications. In *Proceedings of Supercomputing '21 (SC '21).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

1 PROBLEM AND PROPOSED SOLUTION

Portability and resilience are growing problems in high performance computing with different requirements that do not always get along. More systems are relying on accelerators for computational power while memory hierarchies expand to feed said accelerators. As supercomputers increase in scale and heterogeneity, efficiently leveraging the hardware becomes more difficult and the mean time between failures decreases. Portability abstractions such as Kokkos [1] address the heterogeneity problem while checkpoint/restart systems such as VeloC [4] provide resilience against system failures. We present a proof of concept that augments Kokkos applications using VeloC by enabling access pattern aware checkpointing (i.e., by leveraging application and system awareness to create efficient portable checkpoints).

SC '21, November 14–19, 2021, St Louis, MO

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

Developing optimized application specific checkpoint systems that fully leverage both application and system characteristics is not practical and certainly not portable across applications. State of the art checkpoint systems can automate or simplify checkpointing but also introduce important constraints. Some checkpoint systems do not address all the challenges of checkpointing, leaving the application developer or user to handle the rest. For example, not all checkpoint systems are designed to support accelerator-based systems where data must be explicitly transferred to the host CPU before checkpointing. Existing checkpoint systems such as SCR [2] also require boilerplate code for setup and marking data to checkpoint; some systems provide helpful checkpoint primitives but still require the user to code how the checkpoint is performed.

The portability and checkpoint challenges present an interesting opportunity to leverage information from portability abstractions to augment checkpoint runtimes. Kokkos data abstractions contain information on how data is accessed. For example, whether concurrent accesses with many threads are done through atomics or data replication. These abstractions also simplify detecting more complex access patterns, such as sparse updates. The Kokkos Resilience [3] project combines Kokkos abstractions with state-of-the-art checkpoint runtimes to provide checkpoint capabilities to Kokkos applications while remaining portable and minimizing constraints imposed on the developer. Kokkos Resilience uses VeloC, a scalable asynchronous checkpoint runtime, as the back-end.

There are eight stages for checkpointing. They are: (i) decide checkpoint frequency; (ii) mark regions/data for checkpointing; (iii) synchronize processes and threads; (iv) calculate and allocate buffers for the checkpoint; (v) move data into the correct memory space; (vi) gather data for checkpoint; (vii) use I/O middleware effectively; and (viii) write checkpoint to file. Using Kokkos Resilience, the developer only needs to worry about Stages (i)-(ii). Information from Kokkos abstractions allows Kokkos Resilience to automatically manage memory and removes the need for extensive boiler plate code. Currently, Kokkos Resilience abstractions and VeloC do not support mechanisms that leverage application and system awareness to optimize checkpointing. This gap can result in performance degradation due to I/O bottlenecks and wasted resources. We address the gap with access pattern aware checkpointing.

This poster serves as the groundwork for integrating access pattern aware strategies into Kokkos Resilience abstractions and VeloC. Exploiting this opportunity will allow for more intricate checkpoint optimizations without the complexity normally required to implement and use them. These optimizations would also be portable and accessible by any Kokkos application. Specifically, we demonstrate the effectiveness of access pattern aware checkpointing by leveraging incremental checkpointing for sparsely updated data. We show

how this approach allows for smaller and faster checkpoints while reducing complexity to users. Graph applications such as our case study stand to benefit from incremental checkpoints.

2 USE CASE: GRAPH APPLICATION FIDO

Fido is a graph alignment application for comparing different graphs and serves as our case study for this proof of concept. Fido aligns and compares vertices between graphs by creating a signature for each vertex. The vertex signature is created by generalizing the concept of vertex degree. For each vertex, Fido computes a vertex's graphlet degree vector (GDV) by counting the different substructures it touches. The GDVs are collected into a 2D matrix that is used to align vertices between graphs. Calculating the GDVs requires Fido to iterate over all possible combinations of neighboring vertices within a certain distance. Not all combinations result in valid subgraphs, resulting in a very sparse update pattern that depends heavily on the structure of the input graph. The unpredictable, sparse update pattern opens up opportunities for more efficient checkpoint strategies.

3 PATTERN-AWARE CHECKPOINTING

There are multiple factors that should trigger efficient checkpoints and are ignored by the common checkpoint strategies. These factors can be used to reduce checkpoint size or control checkpoint frequency. Such factors include sparse data updates, distributed data, and communication patterns. Using these factors to guide checkpointing requires application-level pattern awareness. The checkpoint system must know what patterns are present before using specific optimizations. For example, sparse update patterns can trigger a switch from full checkpoints to incremental checkpoints that only save changes since the previous checkpoint.

For Fido, the sparsely updated GDVs trigger incremental checkpoints. Kokkos Resilience would detect sparse update patterns by counting how many entries have been changed since the previous checkpoint. If the count exceeds a certain threshold, then the data is marked for incremental checkpointing and the appropriate calls are made to VeloC. VeloC would then perform the incremental update.

As a first step we implement sparse update detection and incremental checkpointing in Fido to demonstrate that pattern aware checkpointing can reduce checkpoint size and overhead. The initial implementation uses Kokkos abstractions and can be integrated into Kokkos Resilience easily. Sparse update detection scans the data for updates and stores the differences in a map. If there are many updates we make a full checkpoint by copying the data into a buffer and writing the buffer to file using standard I/O. Otherwise, the map is serialized and written to file.

4 RESULTS AND FUTURE WORK

We ran tests on two different sets of graphs with different features and sizes (see Table 1) on a Power9 system using 8 MPI ranks and 4 threads per rank. Each rank makes its own checkpoint independently whenever it finishes a preset number of iterations. The two sets of graphs look at different scientific domains, ecology and road networks. The ecology graphs are smaller graphs with similar regular patterns. The road network graphs are larger, less dense, and have more dissimilar structures. The larger road networks highlight how large checkpoints can overwhelm system I/O.

Set	# of vertices	# of edges	# of iterations	Checkpoint interval
ecology1	1,000,000	4,996,000	6,154M	100M
ecology2	999,999	4,995,991	6,154M	100M
asia_osm	11,950,757	25,423,206	4,724M	100M
germany_osm	11,548,834	24,738,362	3,126M	100M

Table 1: Input graphs for Fido.

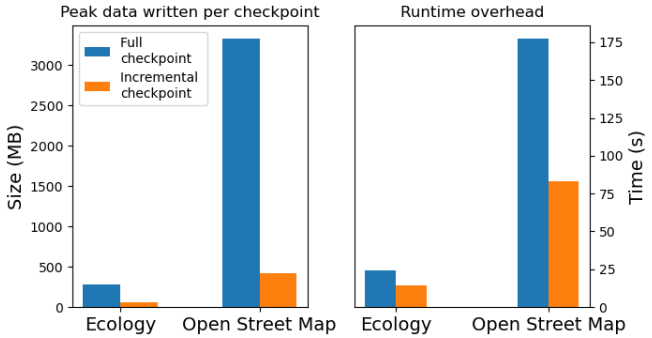

Figure 1: Peak memory written by a single process (left) and runtime overhead (right) for full and incremental checkpoint strategies.

We measure two metrics of interest: the peak data written per checkpoint and the runtime overhead. The peak data written indicates the maximum amount of memory written for any checkpoint. Using full checkpoints as a baseline, Figure 1 shows that incremental checkpoints write 4-8x less data per checkpoint. Smaller checkpoints require less I/O bandwidth and alleviate I/O bottlenecks. The runtime overhead metric highlights how checkpoint size affects the applications overall runtime. The overhead for incremental checkpoints is 47-58% the overhead of full checkpoints. Faster checkpoints can improve application efficiency or enable more frequent checkpoints for more resilience.

In future work, we will extend Kokkos Resilience to automatically detect different access patterns and pass the pattern information to checkpoint systems such as VeloC for pattern aware checkpointing. We will also study more access patterns, optimizations, and factors for efficient checkpoints.

ACKNOWLEDGMENTS

Nigel Tan thanks the advisors on this project: Bogdan Nicolae, Keita Teranishi, Nicolas Morales, Sanjukta Bhowmick and Michela Taufer.

REFERENCES

[1] H. C. Edwards, C. R. Trott, and D. Sunderland. 2014. Kokkos: Enabling Manycore Performance Portability through Polymorphic Memory Access Patterns. *J. of Parallel and Distributed Computing* 74, 12 (2014), 3202 – 3216.

[2] A. Moody, G. Bronevetsky, K. Mohror, and B. R De Supinski. 2010. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proc. of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 1–11.

[3] N. Morales, K. Teranishi, B. Nicolae, C. Trott, and F. Cappello. 2021. Towards High Performance Resilience using Performance Portable Abstractions. In *Proc. of the 27th Int. European Conference on Parallel and Distributed Computing (EuroPar)*.

[4] B. Nicolae, A. Moody, E. Gonsiorowski, K. Mohror, and F. Cappello. 2019. VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale. In *Proc. of 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 911–920.