Reversible debuggers are useful tools for developing and deploying modern applications. However, due to their high memory requirements and runtime overhead, their functionality is generally reserved for rare cases (e.g., identifying short-term memory corruption). This paper describes an alternative approach for implementing a low-overhead memory snapshotting mechanism using fully persistent data structures. Memory usage and performance analyses will be presented and compared against alternative implementations.

**Goal(s)/RQs**

1. Can overhead for Alice be pushed down to ~20% of wall-clock time while still supporting fully reversible execution for unmodified multi-threaded userspace applications?

2. Can we support fully reversible execution for Alice with less than ~20% space overhead (as compared to uninstrumented execution) for long-running applications?

**Background/References**

- Fully persistent data structure: Immutable data structure that returns a copy of itself when modified. All copies retain the same algorithmic complexity guarantees (i.e., older versions don’t get slower).
- Fully reversible execution: Execution of a program that can move forwards and backwards, can be modified in a previous state, and then run forward (supports divergent re-execution).

**Data structure of choice is persistent bit-partitioned hash trie with hash-consing. This allows us to identify duplicate sub-tries, and provies O(log32(n)) updates.**

**Results**

- Our debugger currently supports reverse breakpoints, reverse-step, reverse-continue, as well as normal forward debugging supported by GDB
- After reverse-step, the state of the program can be modified and re-executed.
- Performance results are forthcoming

**Future Work/Acknowledgments**

- Extending our debugger to support multithreaded applications
- Can our library be used to implement mult-shot continuations, transactional memory, backtracking search?
- Integrate into QEMU: Would provide full-system reversible debugging