# Proposal: Efficient Deterministic Execution Runtime

Author One, Author Two
{author1,author2}@ku.edu
University of Kansas, USA

## I. Project Description

Nowadays, shared memory multiprocessors (SMP) are becoming more and more popular in the commodity systems and software programmers are expected to write multi-threaded programs to fully utilize available processor cores. However, writing multi-threaded programs are much more difficult than sequential programs because interleaved accesses to shared memory by multiple threads may result in unanticipated outcome even though inputs are the same. The bugs, caused by memory interleaving, are particularly problematic since it is very difficult to diagnose and reproduce; sometimes even days of stress test fail to manifest the same problem [3].

Many methods are developed to ease the burden of programmers to detect, reproduce, and avoid the interleaving problem of multi-threaded programs. One recent approach is deterministic execution [4], [2]. The key idea of this approach is to completely eliminate the interleaving problem by providing a deterministic global order—using *logical time*—of all shared accesses. It significantly reduces state space which then would result in lower verification cost. Enforcing a global order, however, usually comes at the cost of increased execution time. While existing solutions have shown reasonable performance for scientific multithreaded applications in which threads are generally homogeneous and not I/O intensive especially during the multithreaded execution phase. However, performance will degrade significantly for more general multi-threaded applications in which threads are heterogeneous (i.e., each thread performs different code) and use lots of I/O operations because each thread may need to wait for a certain global logical time to be reached.

In this project, we aim to improve deterministic execution performance of general multi-thread applications with unbalanced blocking I/O operations by reducing the difference between the physical time progress and the logical time progress of each thread. To do this, we categorize I/O operations into two groups: deterministic blocking and non-deterministic blocking operations. The examples of deterministic blocking operations are *sleep()* where programmers specify explicit blocking time. In this case, we can improve deterministic execution performance by adding deterministic logical time when the kernel wakes up the blocking thread. The amount of logical time to add is deterministic, which is based on the supplied physical time, and therefore maintain determinism.

Non-deterministic blocking calls (e.g., *read(), write(), poll()*) are, however, more difficult to handle, since the physical execution time of those calls are inherently nondeterministic because it depends on internal operating system state and physical device state. We will explore two possibilities. First, we will experiment a deterministic method to predict proper logical execution time based on system call parameters such as length and timeout. Also, we will experiment a method to introduce limited numbers of non-determinism to improve accuracy of the prediction. For example, we can measure physical blocking time of the call and then map into finite number of predefined logical time. While latter method will introduce non-determinism but the total interleaving space will be smaller than full

**Deliverables:** We will develop an efficient deterministic execution framework with the consideration of the I/O interference. Also, to show the effectiveness of our framework, we will provide performance analysis of some unbalanced parallel benchmarks from the PARSEC benchmark suite [1] and real-world applications such as Apache and Mysql.

## References

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. Technical Report TR-811-08, Princeton University, January 2008.

[2] J. Devietti, B. Lucia, L. Ceze, and M. Oskin. DMP: deterministic shared memory multiprocessing. In *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 85–96. ACM, 2009.

[3] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. *ACM SIGARCH Computer Architecture News*, 36(1):329–339, 2008.

[4] M. Olszewski, J. Ansel, and S. Amarasinghe. Kendo: efficient deterministic multithreading in software. In *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 97–108. ACM, 2009.