Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

PARALLEL SIMULATION-BASED PREDICTION IN DISCRETE-TIME DISCRETE-STATE-SPACE MARKOV CHAINS WITH GPU IMPLEMENTATION

Yifu Tang[1], Peter W. Glynn[2], and Zeyu Zheng[1]

[1]Dept. of Industrial Eng. and Operations Research, University of California, Berkeley, CA, USA
[2]Dept. of Management Science and Engineering, Stanford University, Stanford, CA, USA

ABSTRACT

Effectively predicting the future performance of a stochastic system via simulation is a critical need. In particular, a challenging class of tasks arises when the stochastic system has an underlying unobservable Markov chain that drives the dynamics, whereas one only gets to partially observe through a function of the underlying Markov process. Previous literature proposed a long-run estimator to predict the future performance system of a stationary system given currently observed states. However, it can be challenging to run a long-run estimator in parallel. In this work, we propose a modified estimator that is easier to simulate in parallel, especially in a way that fits parallel simulation via Graphics Processing Units (GPUs). We discuss implementation procedures on GPUs and then analyze the asymptotic convergence behavior of the parallel estimator and its corresponding central limit theorem.

1 INTRODUCTION

In a range of operational problems, one is interested in predicting the future performance of a system given currently observed states. These problems arise naturally in applications including supply chain management, manufacturing operations, and service systems. Lim and Glynn (2023) developed a simulation-based framework for constructing predictors of future performance measures when the complete system state is not directly observable. In their model, it is assumed that there is an underlying S-valued Markov chain $(X_u : u \geq 0)$ that is not observable. The observable process is denoted as $(Y_n = f(X_n) : n \geq 0)$ for some function $f : S \to \mathbb{R}^d$. With this observable model, one is interested in the performance $Z_n = g(X_n)$ for some performance function $g : S \to \mathbb{R}^d$ in future states. In this paper, we consider a special case of Lim and Glynn (2023) in which the future prediction we are interested in and the observable quantities are the same, i.e., $f(x) = g(x)$, for all $x \in S$ (and therefore $Z_n = Y_n$ for all $n \geq 0$).

Specifically, let $(X_u : u \geq 0)$ be a Markov process in a state space S with transition matrix $P = (P(x,y) : x,y \in S)$ and time index $u \in \mathbb{Z}_{\geq 0}$. In our model, $(X_u : u \geq 0)$ denotes the sequence of underlying states that is not observable. Assume that $(X_u : u \geq 0)$ is positive recurrent and π, a probability distribution on S, is the steady-state distribution. Furthermore, there is a function $f : S \to \mathbb{R}^d$, and an observable process $(Y_u : u \geq 0)$ defined as $Y_u = f(X_u), u \in \mathbb{Z}_{\geq 0}$. Then, the problem can be viewed as, given $y \in \mathbb{R}^d$ and a fixed number $t \geq 0$, one is interested in estimating

$$k_t(y) := \mathbb{E}[Y_{u+t} | Y_u = y, X_0 \sim \pi],$$

given that X_0 follows the steady-state distribution π.

By the formula of conditional expectation, we have

$$k_t(y) = \sum_{x \in S} \mathbb{E}[f(X_{u+t}) | X_u = x] \mathbb{P}(X_u = x | Y_u = y).$$

As stated in Lim and Glynn (2023), there are some challenges to estimations of $k_t(y)$:

1. When the state space S is large, simulating the Markov chain $(X_u : u \geq 0)$ may rarely see $f(X_u) = y$;
2. Since $f : S \to \mathbb{R}^d$ may compress information, it is likely not to be injective. Therefore, the conditional distribution $\mathbb{P}(X_u = x | Y_u = y)$ is not trivial to determine.

To overcome the challenges, simulation-based prediction is proposed in Section 4 of Lim and Glynn (2023). In the discrete-time Markov process setting, to construct the estimator of $k_t(y)$, one runs a long path of the Markov process $X_0, X_1, \cdots, X_{n-1}$. For each X_i where $i = 0, 1, 2, \cdots, n-1$, if $Y_i = f(X_i) = y$, then, one simulate $m-1$ independent paths starting at X_i. Namely, we perform $m-1$ independent simulations of the Markov chain $(\hat{X}_u : \hat{X}_0 = X_i, u \geq 0)$, where its transition kernel is still P. Denote $X_{i1}, \cdots, X_{i,m-1}$ to be the $m-1$ outcomes of our simulations of $\hat{X}_t$ (we name these $m-1$ paths "sub-paths") and $Y_{ij} = f(X_{ij})$ for $j = 1, 2, \cdots, m-1$. Then, an estimator of $k_t(y)$ in Lim and Glynn (2023) is

$$k_n(y) = \frac{\sum_{i=0}^{n-1} (\frac{1}{m} Y_{i+t} + \frac{1}{m} \sum_{j=1}^{m-1} Y_{ij}) I(Y_i = y)}{\sum_{i=0}^{n-1} I(Y_i = y)}. \tag{1}$$

This estimator is shown to be asymptotically unbiased and enjoys valid central limit theorems, see Theorems 3 and 4 in Lim and Glynn (2023). However, when simulating $k_n(y)$, it requires sequentially simulating a long path $X_0, X_1, \cdots, X_{n-1}$. Therefore, the time complexity of computing $k_n(y)$ is $O(nm)$. When n is large, simulating $k_n(y)$ requires a computational time that is linearly increasing with respect to n.

Therefore, in this work, we aim at giving a parallel estimator for $k_t(y)$ that can be implemented on any parallel computing device, and especially on Graphics Processing Units (GPUs). The discussion of the parallel estimator is in Section 3.

In section 3.1, we develop a modification of the long-run average estimator $k_n(y)$ that makes it easier to compute in parallel. Our basic idea comes from the regenerative method for simulating the steady-state of a Markov chain, see Chapter IV, Section 4 of Asmussen and Glynn (2007). Our parallel estimator is then a modification of the long-run average estimator $k_n(y)$. The key idea is to modify the long-run average estimator $k_n(y)$ so that it averages over independent regenerative paths. The regenerative paths start at the same initial state $z \in S$ and terminate at z as well. Denoting $\tau = \inf\{t > 0 : X_t = z\}$. Since the regenerative paths are independent of each other, we can simulate the paths in parallel.

In section 3.2, we discuss the implementation of our parallel estimator on parallel computing devices, especially on Graphics Processing Units (GPUs). Our parallel estimator can naturally be implemented on fully independent parallel processors, such as multi-core CPUs, where each processor is operated by its own controller. We further show that our parallel estimator can be implemented on parallel computing devices without independent parallel processors, such as Graphics Processing Units (GPUs). A GPU is a special type of parallel computing device that is designed to implement similar tasks in parallel. The detailed comparison of the two types of parallel processors is in Section 3.2, as well as the GPU computing model.

In section 3.3, the asymptotic behavior of our modified estimator is analyzed. We will use a somewhat different approach to our modified estimator as compared to that in Lim and Glynn (2023) due to several reasons. First, our modified estimator is not a precise splitting of $k_n(y)$ into regenerative paths. "Boundary terms" may arise in each simulation cycle. Because of such "boundary terms", the concatenation of regenerative cycles may be different from the long-run average paths (see detail arguments at the end of Section 3.1). Secondly, the regenerative paths are independent of each other in our modified estimator. Therefore, the probabilistic tools for our analysis will be different from Lim and Glynn (2023). Specifically, we show asymptotic unbiasedness and also the central limit theorem for our proposed estimator.

In Section 4, we will implement our parallel estimator on the GPU and compare it to the CPU implementation of $k_n(y)$ in Eq. (1). We show the accuracy and implementation time of the estimators on the CPU and the GPU, illustrating that one can take advantage of using the GPU as a parallel computing device to implement our parallel estimator.

2 THE MODEL, NOTATIONS, AND ASSUMPTIONS

We follow the model in Lim and Glynn (2023). Let $(X_u : u \geq 0)$ be a Markov process in state space S, and $u \in \mathbb{Z}_{\geq 0}$ be the time index. We assume the state space S is finite or countably infinite throughout this paper.

In the model, $(X_u : u \geq 0)$ denotes the underlying states that is not observable. However, we have access to the transition matrix $P = (P(x,y) : x,y \in S)$ of the Markov process $(X_u : u \geq 0)$. We assume that the steady state distribution of $(X_u : u \geq 0)$ is unique and is denoted as π.

There is a function $f : S \to \mathbb{R}^d$, and an observable process $(Y_u : u \geq 0)$ defined as $Y_u = f(X_u), u \in \mathbb{Z}_{\geq 0}$. Since f may not necessarily be injective,

$$f^{-1}(y) \triangleq \{x \in S : f(x) = y\},$$

may have multiple elements. We assume throughout the paper $f^{-1}(y) \neq \varnothing$.

Then, the problem can be viewed as, given $y \in \mathbb{R}^d$ and a fixed integer $t > 0$, one is interested in estimating

$$k_t(y) := \mathbb{E}[Y_{u+t} | Y_u = y, X_0 \sim \pi], \tag{2}$$

where $X_0 \sim \pi$ denotes that X_0 follows the steady-state distribution π.

We introduce the following notation in this paper. For two random variables X and Y, we write $X \overset{d}{=} Y$ if X and Y has the same distribution. Let $\lfloor x \rfloor$ denote the largest integer less than or equal to $x \in \mathbb{R}$, while $\lceil x \rceil$ denotes the smallest integer that is not less than $x \in \mathbb{R}$. Denote $A_n \xrightarrow{a.s.} A$ or $\lim_{n\to\infty} A_n \overset{a.s.}{=} A$ that A_n converges to A almost surely in probability space induced by the Markov chain $(X_u : u \geq 0)$. And we denote $A_n \overset{D}{\Rightarrow} A$ or $\lim_{n\to\infty} A_n \overset{d}{=} A$ that A_n converges to A in distribution, see Chung (2001).

3 THE PARALLEL ESTIMATOR AND ITS GPU IMPLEMENTATION

In this section, we proposed the modified parallel estimator of (2). Then, we discuss the implementation of the parallel estimator on general parallel computing devices. After that, we show, explicitly, the way to implement the parallel estimator on the GPU. Throughout this section, we assume that the Markov Chain $(X_u : u \in \mathbb{Z}_{\geq 0})$ is a discrete-time Markov chain.

3.1 The Parallel Estimator

We assume that $\mathbb{P}(f(X) = y | X \sim \pi) > 0$ for a given y. With this assumption, the long-run average estimator in (1) will have a positive probability that $I(Y_i = y)$. In this setting, given an integer $t > 0$, we are interested in estimating

$$k_t(y) = \mathbb{E}[f(X_{u+t}) | Y_u = y, X_0 \sim \pi] = \sum_{x \in S} \mathbb{E}[Y_{u+t} | X_u = x] \mathbb{P}(X_u = x | Y_u = y, X_0 \sim \pi).$$

To modify the estimator $k_n(y)$ in (1) to be easier for parallel computing, the main idea is to split a long-run path into regenerative cycles, similar to the idea in Chapter IV, Section 4 of Asmussen and Glynn (2007). Specifically, with the assumption that the Markov chain $(X_u : u \geq 0)$ is positive recurrent, for any $X_0 = z \in S$, defining $\tau = \inf\{s > 0 : X_s = z\}$, then $\tau < \infty$ almost surely. We aim at modifying the numerator and denominator of $k_n(y)$ into a sum of M independent regenerative paths, where M is a pre-determined fixed number. Since the regenerative cycles are independent of each other, it is easier for us to simulate them in parallel.

In each simulation path $X_0^p = z, X_1^p, \cdots, X_{\tau_p - 1}^p, X_{\tau_p}^p = z, \cdots, X_{\tau_p + t - 1}^p$, whenever $X_i^p (0 \leq i \leq \tau_p - 1)$ satisfies that $f(X_i^p) = y$, one start simulating m independent sub-paths that start at X_i^p, namely $(\tilde{X}_u^{j,p}, u \geq 0)$, given $\tilde{X}_0^{j,p} = X_i^p$ and its transition kernel being P, the same as $(X_u : u \geq 0)$. For each $j = 1, 2, \cdots, m$, denote

$X_{ij}^{p} := \tilde{X}_{t}^{j,p}$. Then, our parallel estimator of $k_t(y)$ is

$$\text{(Parallel Estimator)}\quad \hat{k}(y;M,m,z) = \frac{\sum_{p=1}^{M}\sum_{i=0}^{\tau_p-1}[\frac{1}{m}f(X_{i+t}^{p}) + \frac{1}{m}\sum_{j=1}^{m-1} f(X_{ij}^{p})]I(f(X_i^{p}) = y)}{\sum_{p=1}^{M}\sum_{i=0}^{\tau_p-1} I(f(X_i^{p}) = y)}. \tag{3}$$

Our parallel modified estimator $\hat{k}(y;M,m,z)$ and $k_n(y)$ are not exactly equivalent. If one chooses $\hat{k}(y;M,m,z)$ and $\hat{k}_n(y)$ starting at the same initial state $X_0 = z \in S$, and the terminating time n of $k_n(y)$ as a returning time to the initial state z, the major part of the two estimators are equivalent. However, there are subtle differences between the direct concatenation of different regenerative paths in $\hat{k}(y;M,m,z)$ and the original long-run average estimator $k_n(y)$. In each regenerative cycle $X_0^{p},\cdots,X_{\tau_p-1}^{p}$, if one has $f(X_j^{p}) = y$ for some $j \geq \tau_p - t + 1$, then, one will simulate $Y_{j+t}^{p} = f(X_{j+t}^{p})$ in this cycle, which is beyond the returning time τ_p. Such Y_{j+t} terms can be viewed as the "boundary terms" of a regenerative cycle. Since different regenerative paths are independent of each other, the "boundary terms" Y_{j+t}^{p} in regenerative path p are independent of other regenerative paths $(X_k^{q} : k \geq 0)$ for $q \neq p$. However, denoting $0 = t_0 < t_1 < \cdots < t_l \leq n < t_{l+1}$ in $k_n(y)$ as the returning times to the initial state $X_0 = z \in S$, such "boundary terms" Y_{j+t} (for $t_r - t < j \leq t_{r+1}$) will appear in the next regenerative cycle when $f(X_j) = y$. Here t_l is the l-th returning time to the initial state z such that $t_l \leq n < t_{l+1}$, and t_r is the r-th returning time such that $t_r - t < j \leq t_{r+1}$. In this case, different regenerative cycles in $k_n(y)$ may fail to be independent of each other due to such "boundary terms" of $k_n(y)$.

We provide the implementation of $\hat{k}(y;M,m,z)$ in parallel computing devices (including GPUs) in Section 3.2. We discuss the asymptotic behavior of $\hat{k}(y;M,m,z)$ in Section 3.3.

3.2 Implementation of the Parallel Estimator on Graphics Processing Units

Clearly, our parallel estimator can be implemented on any group of fully independent parallel processors. By fully independent, we mean that each processor is capable of executing distinct tasks with their own controller and memory. An example of fully independent parallel processors is a multi-core CPU system, where each processor is operated by its controlling operator. On such fully independent parallel processors, one can assign the M different regenerative cycles simulation tasks to different processing units flexibly. For example, if there are k fully independent parallel processors and we aim at simulating M tasks. When $k < M$, one can first assign k tasks to the k processors and make them implement the tasks in parallel. Whenever one processor finishes its tasks, one can assign it a new task that has not been completed. This is a naive but flexible way of implementing tasks on fully independent parallel processors.

However, Graphics Processing Units (GPUs) do not constitute a group of fully independent processors. Although GPUs contain many parallel computing units, these units are coordinated by a shared controller, which limits them to executing highly similar tasks. Similarly, other devices with many processors at a comparable price to GPUs often organize the processors under a shared controller, thus lacking full independence.

To better understand the benefit of GPU computing for some tasks, we next introduce the GPU parallel computing model. Modern GPUs (such as NVIDIA GPUs, see NVIDIA Corporation (2022)) adopt a two-level hierarchical structure: several processing cores form a unit called a Streaming Multiprocessor (SM), and multiple Streaming Multiprocessors together comprise the GPU. All cores within the GPU share access to memory, which facilitates rapid data exchange and accelerates computation. A thread is the basic unit of execution, running a single sequence of instructions on one core. A group of threads executing within the same Streaming Multiprocessor is referred to as a thread block. Each thread is uniquely identified by two indices: its position within its thread block and the position of the block within the entire GPU.

To distribute tasks to the threads, one defines a kernel function, which specifies the computation each thread performs. Although each thread executes the same kernel function, threads can behave differently by accessing different portions of memory based on their thread and block indices. This design enables

Algorithm 1: This algorithm shows the parallel implementation of estimator $\hat{k}(y;M,m,z)$ on GPU.

```
Input: Number of threads (Parallel Computing Units) available $k$, number of sub-path $m$, initial
       state $z \in S$, transition kernel $P$, future time $t$, conditional state $y$, total replication number $M$
Transfer Random Number Generating states from CPU to GPU;
for Thread number $d = 1,2,\cdots,k$, in parallel do
    for $p = 1,2,\cdots,\lceil \frac{M}{k} \rceil$, on each thread, sequentially do
        Initialize $X_0^p = z$;
        Denote current state $x = z$;
        Simulate next state of the initial state $w \sim P(x,\cdot)$;
        Set $F_p^d, N_p^d := 0$, denoting the cumulative quantity of the numerator and denominator;
        Set $i := 0$, recording the length of path;
        while $w \neq z$ do
            Update $x \leftarrow w$;
            Update index $i \leftarrow i+1$ and $X_i^p := w$;
            if $f(x) = y$ then
                for $j = 1,2,\cdots,m-1$ do
                    Let $X_0^j = x$;
                    Simulate the Markov chain $(X_u^j : u \geq 0)$, obtaining the result $X_0^{[j]},\cdots,X_t^{[j]}$;
                end
                Update $F_p^d \leftarrow F_p^d + \frac{m-1}{m}\sum_{j=1}^{m-1} f(X_t^{[j]})$;
            end
            Update $N_p^d \leftarrow N_p^d + 1$;
            If $f(X_{i-t}) = y$, then update $F_p^d \leftarrow F_p^d + \frac{1}{m} f(X_i^p)$;
            Update $w \sim P(x,\cdot)$ ;
        end
        Simulate $X_{\tau_p+1}^p,\cdots,X_{\tau_p+t-1}^p$;
        for $j = 0,1,2,\cdots,t-1$ do
            If $f(X_{\tau_p+j-t}^p) = y$, update $F_p^d \leftarrow F_p^d + \frac{1}{m} f(X_{j+\tau_p}^p)$
        end
        Return $F_p^d$ and $N_p^d$;
    end
end
Transfer data $\{F_p^d, N_p^d\}_{p=1,2,\cdots,\lceil \frac{M}{k} \rceil, d=1,2,\cdots,k}$ back to CPU;
Construct $\hat{k}(y;M,m,z) = \frac{\sum_{p,d} F_p^d}{\sum_{p,d} N_p^d}$ on CPU;
Output: The final estimator $\hat{k}(y;M,m,z)$
```

GPUs to efficiently perform a large number of similar operations on different data simultaneously. See Owens et al. (2008) and NVIDIA Corporation (2022) for more details on GPU computational models.

Although GPUs are powerful in parallel computing, computing processes cannot be implemented directly on GPUs. One will need Central Processing Units (CPU) to control and distribute the tasks and to transfer data to GPU. The data-transferring process between the CPU and the GPU is more time-consuming than that in a single GPU or a single CPU. This means that GPUs may not take much benefit to implement tasks that require data transfer. We refer to Kirk and Wen-Mei (2016), Ryoo et al. (2008), and Nickolls et al. (2008) for more details about the benefits of GPU programming.

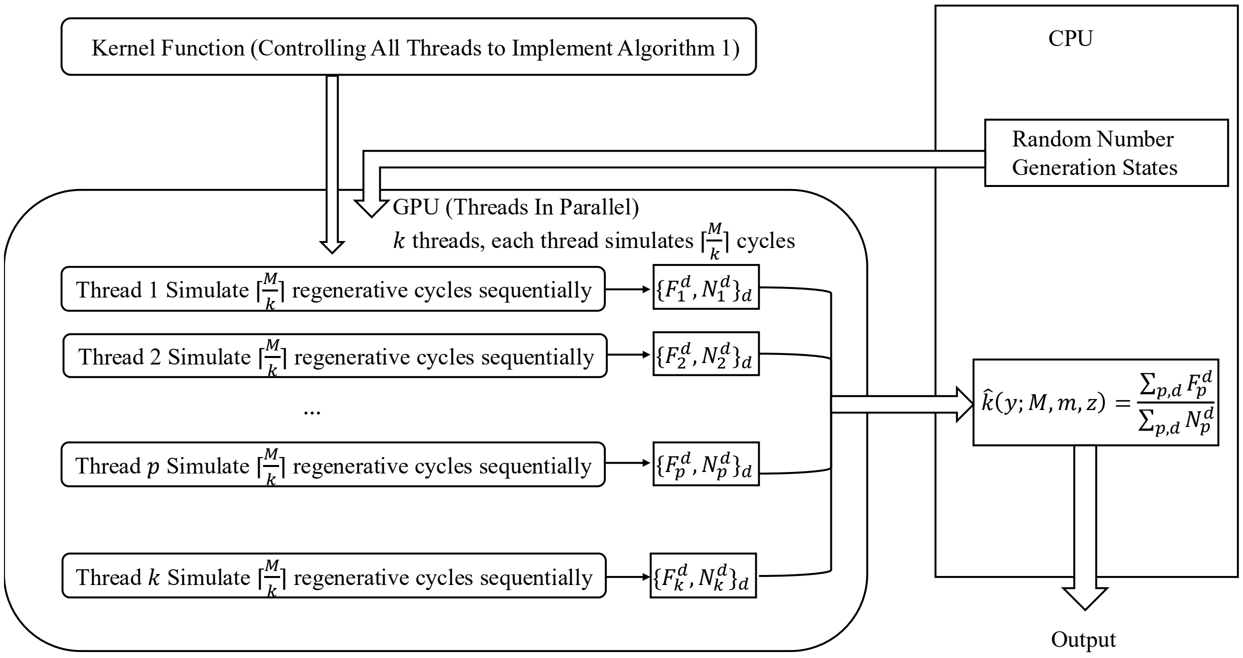
Therefore, GPUs are good at performing computational jobs that require implementing massive similar tasks but include few communications between each parallel unit. Because regenerative simulation involves repeated, structurally similar, and independent simulations across different paths, the GPU's architecture is highly suited for implementing our algorithm.

With the above characteristics of GPU computational structure, we next introduce the GPU implementation of $\hat{k}(y;M,m,z)$. Let there be k available threads on GPU for our parallel computing. We first create a random number generation (RNG) state on the CPU. Then, we transfer the RNG states from the CPU to the GPU's memory, assigning each thread a random number generation state. The kernel function then controls each thread on the GPU to simulate $\lceil \frac{M}{k} \rceil$ independent regenerative cycles sequentially. We denote the terms in the summation of the numerator and the denominator of $\hat{k}(y;M,m,z)$ to be

$$\begin{cases} F_p(y,z,m) := \sum_{i=0}^{\tau_p-1}[\frac{1}{m}f(X_{i+t}^p)+\frac{1}{m}\sum_{j=1}^{m-1}f(X_{ij}^p)]I(f(X_i^p)=y), \\ N_p(y,z,m) := \sum_{i=0}^{\tau_p-1}I(f(X_i^p)=y). \end{cases} \tag{4}$$

Then, in each thread, $\lceil \frac{M}{k} \rceil$ different $F_p(y,z,m)$ and $\lceil \frac{M}{k} \rceil$ different $N_p(y,z,m)$ will be computed and stored in the GPU's memory during simulation.

After all the threads have completed their own tasks, the result in the GPU's memory will then be sent back to the CPU's memory and one constructs the final result $\hat{k}(y;M,m,z)$ on the CPU. Specifically, the implementation of $\hat{k}(y;M,m,z)$ on a GPU with k parallel computing units available is summarized in Algorithm 1. The process is illustrated in Figure 1.



Kernel Function (Controlling All Threads to Implement Algorithm 1)
CPU
Random Number Generation States
GPU (Threads In Parallel)
k threads, each thread simulates $\lceil \frac{M}{k} \rceil$ cycles
Thread 1 Simulate $\lceil \frac{M}{k} \rceil$ regenerative cycles sequentially
$\{F_1^d, N_1^d\}_d$
Thread 2 Simulate $\lceil \frac{M}{k} \rceil$ regenerative cycles sequentially
$\{F_2^d, N_2^d\}_d$
...
Thread p Simulate $\lceil \frac{M}{k} \rceil$ regenerative cycles sequentially
$\{F_p^d, N_p^d\}_d$
Thread k Simulate $\lceil \frac{M}{k} \rceil$ regenerative cycles sequentially
$\{F_k^d, N_k^d\}_d$
$\hat{k}(y;M,m,z) = \frac{\sum_{p,d} F_p^d}{\sum_{p,d} N_p^d}$
Output

Figure 1: The figure illustrates the GPU Implementation Process of Computing $\hat{k}(y;M,m,z)$.

Remark 1 In our assignment method on the GPU, the time it takes for each thread to complete is random, since the cycle length is random. This will likely cause some threads to terminate first while others are still computing. The threads that complete their tasks first will then wait for all the threads to complete their tasks. However, even if some threads may spend time idle after completing their tasks, our GPU implementation of $\hat{k}(y;M,m,z)$ is still shown to be more efficient than the CPU implementation of $k_n(y)$ in the experiment in Section 4.

3.3 Asymptotic Behavior of the Parallel Estimator

We analyze the asymptotic behavior of the parallel estimator $\hat{k}(y;M,m,z)$ in this section. In Lim and Glynn (2023), they have shown that given some assumptions on the long-run convergence of $I(Y_i = y)$ and Y_i, and integrability of $f(X_i)$, their estimator $k_n(y)$ will exhibit a central limit theorem for which $\sqrt{n}(k_n(y) - k_t(y))$ converges weakly to a Gaussian random variable.

As discussed at the end of Section 3.1, our parallel estimator is different from the direct splitting of $k_n(y)$. We will follow a different framework for analyzing the asymptotic behavior of $\hat{k}(y;M,m,z)$, using different assumptions from Lim and Glynn (2023).

We will first state two Propositions to assist our analysis of the convergence behavior of $\hat{k}(y;M,m,z)$. Proposition 1 is a restatement of Theorem 2.18 from Hall and Heyde (2014) that gives us a tool to analyze the almost sure convergence of our parallel estimator $\hat{k}(y;M,m,z)$. Proposition 2 is the "stochastic version" of the ergodic theorem for our convergence result. We will apply Propositions 1 and 2 to give asymptotic unbiasedness of $\hat{k}(y;M,m,z)$ with different assumptions from those in Lim and Glynn (2023). Finally, we will use the multivariate central limit theorem and the Delta method to give the central limit theorem for $\hat{k}(y;M,m,z)$. We write F_p and N_p short for $F_p(y,z,m)$ and $N_p(y,z,m)$ in Eq. (4) when there is no ambiguity of y,z,m.

Proposition 1 (Strong Law of Large Numbers for Martingales, Restatement of Special case) Let $(S_n = \sum_{i=1}^{n} X_i, \mathscr{F}_n, n \geq 1)$ be a martingale. If for some $p \in [1,2]$, such that

$$\sum_{n=1}^{\infty} \frac{1}{n^p} \mathbb{E}[|X_n|^p | \mathscr{F}_{n-1}] < +\infty,$$

Then, we have $\frac{S_n}{n} \xrightarrow{a.s.} 0$, as $n \to \infty$.

We refer the proof of Proposition 1 to Hall and Heyde (2014). Applying Proposition 1, we have the following proposition.

Proposition 2 (Stochastic Version of Ergodic Theorem) Let $\xi = \xi(\omega, x)$ be a r.v. parameterized by $x \in S$, i.e., for each $x \in S$, there is a probability distribution $\xi | x$. We denote $h(x) = \mathbb{E}[\xi | x]$. Assume that for some $1 < p \leq 2$,

$$\sup_{x \in S} \{\mathbb{E}|\xi(x)|^p + |h(x)|^p\} = C < +\infty.$$

Let X_j be a time-homogeneous Markov chain with steady-state distribution π. Let $Y_j \overset{d}{=} \xi | (x = X_j)$ for each j. Furthermore, $Y_1, Y_2, \cdots$ are independent of each other. Then, we have

$$\lim_{n \to \infty} \frac{1}{n} \sum_{j=0}^{n-1} Y_j \overset{a.s.}{=} \mathbb{E}_{X \sim \pi}[\mathbb{E}[\xi(X)]].$$

Proof of Proposition 2. Let $f(x) = \mathbb{E}[\xi | x]$ and $D_j = Y_j - h(X_j)$. Then, we have

$$\frac{1}{n} \sum_{j=0}^{n-1} Y_j = \frac{1}{n} \sum_{j=0}^{n-1} h(X_j) + \frac{1}{n} \sum_{j=0}^{n-1} D_j.$$

By Ergodic Theorem, see for example Meyn and Tweedie (2009), we have

$$\frac{1}{n} \sum_{j=0}^{n-1} h(X_j) \xrightarrow{a.s.} \mathbb{E}_{X \sim \pi}[h(X)].$$

It remains to analyze $\frac{1}{n}\sum_{j=0}^{n-1} D_j$. We define the filtration $\mathscr{F}_j := \sigma(X_0, X_1, \cdots, X_j, D_0, \cdots, D_j)$ for $j = 0, 1, 2, \cdots$, and $M_n = \sum_{j=0}^{n} D_j$. Then, note that for $n \geq 1$,

$$\begin{aligned}\mathbb{E}[M_n|\mathscr{F}_{n-1}] &= \mathbb{E}[M_{n-1}|\mathscr{F}_{n-1}] + \mathbb{E}[D_n|\mathscr{F}_{n-1}] \\ &= M_{n-1} + \mathbb{E}[D_n|X_{n-1}, \xi_{n-1}] \\ &= M_{n-1} + \sum_{x\in S} \mathbb{E}[D_n|X_n = x]P(X_{n-1}, x) \\ &= M_{n-1}.\end{aligned}$$

Furthermore,

$$\mathbb{E}|M_n| = \mathbb{E}\left|\sum_{j=0}^{n} D_j\right| \leq \sum_{j=0}^{n} \mathbb{E}|D_j|.$$

And for each $j = 0, 1, 2, \cdots$,

$$\begin{aligned}\mathbb{E}|D_j| &= \mathbb{E}|Y_j - h(X_j)| \\ &= \mathbb{E}[\mathbb{E}[|Y_j - h(X_j)||X_j]] \\ &\leq \mathbb{E}[\mathbb{E}[|Y_j||X_j] + |h(X_j)||X_j] \\ &\leq \sup_{X\in S}\{\mathbb{E}|\xi(X)| + |h(X)|\} < \infty.\end{aligned}$$

This implies that $\mathbb{E}|M_n| < +\infty$ for all $n \geq 0$. Therefore, $\{M_n\}$ is a Martingale w.r.t. $\{\mathscr{F}_n\}$. Now, in Theorem 1, take $S_n = M_n$ and $D_n = X_n$. We note that since $p > 1$,

$$\sum_{n=1}^{\infty} \frac{1}{n^p}\mathbb{E}[|D_n|^p|\mathscr{F}_{n-1}] \leq C\sum_{n=1}^{\infty} \frac{1}{n^p} < +\infty.$$

Apply Theorem 1, we conclude that as $n \to \infty$, $\frac{1}{n}M_n \xrightarrow{a.s.} 0$. Therefore, we conclude Theorem 2. □

We next turn to show the asymptotic unbiasedness of the parallel estimator.

Theorem 1 (Asymptotic Unbiasedness) Assume that there is a constant $C > 0$, $\mathbb{E}|\xi(x)|^p < C < \infty$ for any $x \in S$ and some $1 < p < 2$, and $\mathbb{E}\tau^2 < +\infty$. Then we have, for any given integer $m \geq 1$,

$$\lim_{M\to\infty} \hat{k}(y; M, m, z) \overset{a.s.}{=} k_t(y).$$

Proof of Theorem 1. We define stopping times $0 = t_0 < t_1 < \cdots$ as

$$t_j = \inf\{t > t_{j-1} : X_t = z\}, j \geq 1.$$

For any integer integer n, define $l = l(n)$ to be a integer-valued random variable such that $t_l \leq n < t_{l+1}$. Consider $m-1$ independent Markov chains $(\hat{X}_u^{[j]}, u \geq 0)$ starting at $\hat{X}_0 = x$, where $j = 1, 2, \cdots, m$. We let

$$\xi(\omega, x) := \frac{1}{m}\sum_{j=1}^{m-1} f(\hat{X}_t^{[j]})I(f(x) = y).$$

Note that

$$\left|\sum_{j=0}^{n} \xi(X_j) - \sum_{j=0}^{t_l-1} \xi(X_j)\right| \leq \sum_{j=t_l}^{n} |\xi(X_j)| \leq \sum_{j=t_l}^{t_{l+1}} |\xi(X_j)|.$$

Now denote

$$Z_l := \sum_{j=t_l}^{t_{l+1}} |\xi(X_j)|.$$

Since $\mathbb{E}|Z_l|^p < \infty$, we have, for $p > 1$,

$$\sum_{l=1}^{\infty} \mathbb{P}(Z_l \geq \varepsilon l) \leq \sum_{l=1}^{\infty} \frac{\mathbb{E}|Z_l|^p}{\varepsilon^p l^p} < \infty.$$

By Borel-Cantelli Lemma, this implies that $\frac{Z_l}{l} \xrightarrow{a.s.} 0$. Therefore, it implies

$$\frac{1}{n}\left|\sum_{j=0}^{n} \xi(X_j) - \sum_{j=0}^{t_l-1} \xi(X_j)\right| \xrightarrow{a.s.} 0.$$

Then, denoting

$$h(x) := \mathbb{E}\left[I(f(x)=y)\frac{1}{m}\sum_{j=1}^{m-1} f(\hat{X}_t^{[j]})\right],$$

we have, almost surely,

$$\begin{aligned}\lim_{l\to\infty} \frac{1}{t_l}\sum_{j=0}^{t_l-1} \xi(X_j) &= \lim_{n\to\infty} \frac{1}{n}\sum_{j=0}^{n} \xi(X_j) \\ &= \mathbb{E}_{x\sim\pi}[h(x)] \\ &= \int_S \frac{m-1}{m}\mathbb{E}[I(f(x)=y)f(\hat{X}_t)]\pi(dx) \\ &= \frac{m-1}{m}\int_S \mathbb{E}[f(\hat{X}_t)|X_0=x]I(f(x)=y)\pi(dx)\end{aligned} \tag{5}$$

We next analyze the terms

$$Q_p := \frac{1}{m}\sum_{i=0}^{\tau_p-1} I(f(X_i^p)=y)f(X_{i+t}^p), p=1,2,\cdots,M.$$

We aim at finding the asymptotics of $\sum_{p=1}^{M} Q_p$ as $M \to \infty$. It is worth noting that this term is not merely the decomposition of the numerator in the long-run estimator (1). This is because, since the different regenerative cycles are independent, the term $f(X_{i+t}^p)$ may be different from the corresponding term in $(X_u^{p+1}, u \geq 0)$. By renewal reward theory,

$$\frac{\mathbb{E}Q_p}{\mathbb{E}\tau_p} = \frac{1}{m}\int_S \mathbb{E}[f(X_t)|X_0=x]I(f(x)=y)\pi(dx).$$

And by $\mathbb{E}\tau^2 < +\infty$, we have $\mathbb{E}Q_p^2 \leq \mathbb{E}\left[\tau_p^2 \sup_x |f(x)|^2\right] < +\infty$. Then, by the strong law of large numbers, we have, as $M \to \infty$,

$$\frac{1}{M}\sum_{p=1}^{M} Q_p \xrightarrow{a.s.} \frac{\mathbb{E}\tau}{m}\int_S \mathbb{E}[f(X_t)|X_0=x]I(f(x)=y)\pi(dx) \tag{6}$$

This implies that

$$\lim_{M\to\infty} \frac{1}{M}\sum_{p=1}^{M} F_p(y,z,m) \overset{a.s.}{=} \mathbb{E}\tau \int_S \mathbb{E}[f(X_t)|X_0=x]I(f(x)=y)\pi(dx).$$

Apply similar analysis to the denominator $N_p(y,z,m)$, we have

$$\lim_{M\to\infty} \frac{1}{M}\sum_{p=1}^{M} N_p(y,z,m) \overset{a.s.}{=} \mathbb{E}\tau \int_S I(f(x)=y)\pi(dx).$$

Taking the ratio, we conclude the Theorem. □

Remark 2 Theorem 1 implies that for all $m \geq 1$, the parallel estimator $\hat{k}(y;M,m,z)$ is asymptotically unbiased given the assumptions in Theorem 1. However, in the real application, for a fixed number M, the number of sub-paths m will also contribute to the accuracy. This is because in the proof of the Theorem, we have shown that the long-run average of $\xi(\omega,x)$ will converge to the stationary expectation, as shown in (5). However, m can determine the variance of ξ defined in the proof of Theorem 1, thus impacting the overall convergence of the parallel estimator $\hat{k}(y;M,m,z)$.

We next give the Central Limit Theorem to the parallel estimator.

Theorem 2 (Central Limit Theorem for $\hat{k}(y;M,m,z)$) Assume that the covariance matrix of $F_p := F_p(y,z,m)$ and $N_p := N_p(y,z,m)$ exists as Σ, then, there exists $\sigma > 0$, such that when $M \to \infty$,

$$\sqrt{M}(\hat{k}(y;M,m,z) - k_t(y)) \overset{D}{\Rightarrow} N(0,\sigma^2).$$

Proof of Theorem 2. We first give a joint Central Limit Theorem for the numerator and the denominator of $\hat{k}$ and then apply the Delta Method (see for details of Delta Methods, Vaart (2000)). Since $(F_p(y,z,m), N_p(y,z,m))$, $p = 1,2,\cdots,M$, are M independent random vectors. By Multi-variate CLT, we have, as $M \to \infty$,

$$\sqrt{M}\begin{pmatrix} \frac{1}{M}\sum_{p=1}^{M} F_p \\ \frac{1}{M}\sum_{p=1}^{M} N_p \end{pmatrix} \overset{D}{\Rightarrow} \mathcal{N}(\mu,\Sigma),$$

where $\mu = (\mathbb{E}F_p, \mathbb{E}N_p)$ and Σ is the covariance matrix of F_p and N_p. Consider function $g(x,y) = \frac{x}{y}$, and $\nabla g(x,y) = (\frac{1}{y}, \frac{-x}{y^2})$. Note that $\nabla g(\mu) \neq 0$. We apply the Delta method to $g(\sum_{p=1}^{M} F_p, \sum_{p=1}^{M} N_p)$. Then, as $M \to \infty$,

$$\sqrt{M}\left(g(\frac{1}{M}\sum_{p=1}^{M} F_p, \frac{1}{M}\sum_{p=1}^{M} N_p) - g(\mu) \right) \overset{D}{\Rightarrow} \mathcal{N}(0, \nabla g(\mu)^\top \Sigma \nabla g(\mu)),$$

where $\nabla g(\mu) = \left(\frac{1}{\mathbb{E}N_p}, -\frac{\mathbb{E}F_p}{(\mathbb{E}N_p)^2}\right)$, and

$$\nabla g(\mu)^\top \Sigma \nabla g(\mu) = \frac{1}{(\mathbb{E}N_p)^2}\left(\mathbb{E}[F_p^2] - \frac{2\mathbb{E}[N_p F_p]}{\mathbb{E}N_p} + \frac{\mathbb{E}[N_p^2](\mathbb{E}F_p)^2}{(\mathbb{E}N_p)^2} \right).$$

Hence, we completed the proof. □

Remark 3 Since the Markov chain $(X_t : t \geq 0)$ is positive recurrent, the choice of $X_0 = z \in S$ in practice will not change the convergence behavior of our parallel estimator $\hat{k}(y;M,m,z)$. In practice, one can choose any starting state $X_0 = z \in S$ as long as the underlying Markov chain $(X_u : u \geq 0)$ is positive recurrent.

4 NUMERICAL EXPERIMENT

We conduct a preliminary numerical study that compares the performance of the following two situations.

1. NVIDIA RTX 6000 Ada GPU is used for computing $\hat{k}(y;M,m,z)$.
2. An Apple M4 Pro CPU is used for computing the long-run average estimator $k_n(y)$.

We consider an M/M/1 queue with arrival rate λ and service rate μ as the underlying hidden Markov process. Let $(X_u : u \geq 0)$ denote the number of customers in the system. Whether an arrival or a departure occurs, the discrete time index u will be increased by one, and X_u is transferred to the next state.

At each time step, with probability $p = \frac{\lambda}{\lambda+\mu}$, an arrival occurs, increasing the queue size by one; with probability $q = \frac{\mu}{\lambda+\mu}$, a departure occurs, decreasing the queue size by one if the queue is non-empty (otherwise the queue remains at zero). Let $\lambda < \mu$ so that the traffic intensity is $\rho = \frac{\lambda}{\mu}$. Then, the steady-state distribution exists $\pi(x) = (1-\rho)\rho^x$ for $x \in \mathbb{Z}_{\geq 0}$. The observable process $(Y_u, u \geq 0)$ is defined by

$Y_u = f(X_u)$, where $f(x) = \lfloor \frac{x}{10} \rfloor$, mapping the number of customers into integer buckets of size ten. In practice, one can refer to Y_n as the approximate crowded level of some queue. Given a fixed integer $t \geq 0$ and $y \in \mathbb{Z}_{\geq 0}$, our objective is to estimate the conditional expectation

$$k_t(y) = \mathbb{E}[Y_{u+t} \mid Y_u = y, X_0 \sim \pi] = \mathbb{E}\left[\left\lfloor \frac{X_t}{10} \right\rfloor \middle| f(X_0) = y, X_0 \sim \pi \right],$$

based on simulation. It, in practice, can refer to a prediction of the future crowded level of the queue.

In the experiment, we use the limit of the Monte-Carlo Simulation to determine the real value of $k_t(y)$. Specifically, for a large number N, we sample $Z_1, \cdots, Z_N$ as $\pi(x|f(x) = y), x \in S$, the steady-state distribution conditional on $f(x) = y$. Then, we start simulate t steps of the Markov chain $(X_u : u \geq 0)$ independently, where the initial states are at $Z_1, \cdots, Z_N$. Denoting the outcomes at time step t of the paths to be $\tilde{X}_{1,t}, \cdots, \tilde{X}_{N,t}$. Then, by law of large number, $\frac{1}{N}\sum_{j=1}^{N} f(\tilde{X}_{j,t})$ converges to $k_t(y)$ as $N \to \infty$. We take the empirical mean of $\tilde{X}_{1,t}, \cdots, \tilde{X}_{N,t}$, where $N = 2 \times 10^8$, in our numerical experiment, to be the real value of $k_t(y)$. For $N = 2 \times 10^8$, their empirical mean is shown to converge.

A thorough comparison of the computational costs between CPUs and GPUs requires aspects of price, power consumption, hard-ware devices, etc. We refer the readers to Lee et al. (2010) for a more detailed comparison in the GPU computing society. In this experiment, we compare $k_n(y)$ computed on an Apple MacBook Pro M4 CPU (device price around 1500-2000 USD, representing portable devices) and $\hat{k}(y; M, m, z)$ computed on an NVIDIA RTX 6000 Ada GPU (device price around 6000 USD, representing computational servers). The price here reflects partially the computational costs in practice. We report the corresponding execution time and absolute error of the CPU estimator $k_n(y)$ and the GPU estimator $\hat{k}(y; M, m, z)$. The execution time is defined as the time from the beginning of task assignment to the final estimator is output. When doing the GPU implementation of $\hat{k}(y; M, m, z)$, we assign each thread on the GPU device one regenerative cycle to simulate. The result of the experiment is shown in Table 1.

Table 1: Comparison of CPU vs. GPU estimator performance given parameters $\lambda = 3, \mu = 2, y = 1, t = 5$, in both CPU and GPU implementation, float 64 type precision is used.

Parameters	**CPU time (s)**	**CPU error**	**GPU time (s)**	**GPU error**
$n = 10^5, M = 2^{14}, m = 5$	0.38	0.004	0.8	0.038
$n = 10^6, M = 2^{15}, m = 5$	3.81	0.003	0.79	0.008
$n = 10^7, M = 2^{16}, m = 5$	37.01	0.00085	0.81	0.0008

As shown in the experiment, when achieving the same level of accuracy, the estimator implemented on the GPU takes significantly less time than that on the CPU. In the experiment on the GPU, since the terminating time of each regenerative cycle is random, the threads that finish their task first will wait until other threads complete their tasks. Even if some threads may spend their time idling, the experiment still shows that the estimation on the GPU takes significantly less time than computing $k_n(y)$ on the CPU. Besides, the table shows that the GPU estimator converges faster than the CPU estimator. Doubling M will result in exactly twice of regenerative cycles, while multiplying n by 10 may not increase an exact number of regenerative cycles in the CPU estimator $k_n(y)$. Therefore, the convergence speed of the CPU estimator $k_n(y)$ may not be a constant.

5 CONCLUSION AND FUTURE WORK

In this paper, we have discussed the modification of a long-run average estimator $k_n(y)$ in Eq. (1) into a parallel estimator to perform simulation-based prediction in a discrete time and state space setting. We showed how to implement the parallel estimator on Graphics Processing Units (GPUs). Also, we discussed the asymptotic behavior of the modified parallel estimator $\hat{k}(y; M, m, z)$. There are various future directions. The study of assumptions that ensure convergence of $\hat{k}(y; M, m, z)$ is an interesting future direction. Furthermore, the analysis of $\hat{k}(y; M, m, z)$ only gives an order of convergence as $M \to \infty$ and an intuitive explanation of the impact of m on $\hat{k}(y;M,m,z)$. It is worth giving a thorough, quantified discussion of the order of convergence w.r.t both M and m. Indeed, the development of parallel simulation-based prediction also involves many future directions. First of all, it is worth analyzing the cases when the underlying Markov chain has continuous time or continuous state space. Furthermore, in practice, one may face situations where the time of using parallel computing devices is limited. In this case, one will need to terminate the parallel simulation within a given time. We also plan to extend our analysis of the parallel estimator in the setting that the time of using parallel computing devices is limited.

ACKNOWLEDGMENTS

We would love to express our appreciation and thanks to the anonymous reviewers, for their time, comments and suggestions. They are very helpful to the improvement of our manuscript. The third author acknowledges NSF for partial support.

REFERENCES

Asmussen, S., and P. W. Glynn. 2007. *Stochastic Simulation: Algorithms and Analysis*. New York: Springer.

Chung, K. L. 2001. *A Course in Probability Theory*. 3rd ed. San Diego: Academic Press.

Hall, P., and C. C. Heyde. 2014. *Martingale Limit Theory and its Application*. New York: Academic Press.

Kirk, D. B., and W. H. Wen-Mei. 2016. *Programming Massively Parallel Processors: A Hands-on Approach*. San Francisco: Morgan Kaufmann.

Lee, V. W., C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, *et al*. 2010. "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU". In *Proceedings of the 37th Annual International Symposium on Computer Architecture*. June 19th-23rd, Saint-Malo, France, 451–460.

Lim, E., and P. W. Glynn. 2023. "Simulation-Based Prediction". *Operations Research* 71(1):47–60.

Meyn, S., and R. L. Tweedie. 2009. *Markov Chains and Stochastic Stability*. 2nd ed. Cambridge Mathematical Library. Cambridge: Cambridge University Press.

Nickolls, J., I. Buck, M. Garland, and K. Skadron. 2008. "Scalable Parallel Programming with CUDA: Is CUDA the Parallel Programming Model that Application Developers have been Waiting for?". *Queue* 6(2):40–53.

NVIDIA Corporation 2022. *GPU Performance Background User's Guide*. https://docs.nvidia.com/deeplearning/performance/dl-performance-gpu-background/index.html, Accessed: 10th March 2025.

Owens, J. D., M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. 2008. "GPU Computing". *Proceedings of the IEEE* 96(5):879–899.

Ryoo, S., C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-m. W. Hwu. 2008. "Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA". In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. February 20th-23th, Salt Lake City, UT, USA, 73–82.

Vaart, A. W. v. d. 2000. *Asymptotic Atatistics*. Cambridge: Cambridge University Press.

AUTHOR BIOGRAPHIES

YIFU TANG is a PhD Student at the University of California, Berkeley. His email address is yifutang@berkeley.edu.

PETER W. GLYNN is the Thomas Ford Professor in the Department of Management Science and Engineering (MS&E) at Stanford University. He is a Fellow of INFORMS and of the Institute of Mathematical Statistics, has been co-winner of Best Publication Awards from the INFORMS Simulation Society in 1993, 2008, and 2016, and was the co-winner of the John von Neumann Theory Prize from INFORMS in 2010. In 2012, he was elected to the National Academy of Engineering. His research interests lie in stochastic simulation, queueing theory, and statistical inference for stochastic processes. His email address is glynn@stanford.edu and his homepage is https://web.stanford.edu/~glynn/index.html.

ZEYU ZHENG is an associate professor in the Department of Industrial Engineering and Operations Research (IEOR) and the Berkeley Artificial Intelligence Research Lab (BAIR) at the University of California, Berkeley. He received a Ph.D. in Management Science and Engineering (2018), Ph.D. minor in Statistics (2018) and M.A. in Economics (2016) from Stanford University, and B.S. in Mathematics (2012) from Peking University. His email address is zyzheng@berkeley.edu and his homepage is http://zheng.ieor.berkeley.edu.