Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

A PRELIMINARY STUDY ON ACCELERATING SIMULATION OPTIMIZATION WITH GPU IMPLEMENTATION

Jinghai He[1], Haoyu Liu[1], Yuhang Wu[1], Zeyu Zheng[1], and Tingyu Zhu[1]

[1] Dept. of Industrial Eng. and Operations Research, University of California Berkeley, Berkeley, CA, USA

ABSTRACT

We provide a preliminary study on utilizing GPU (Graphics Processing Unit) to accelerate computation for three simulation optimization tasks with either first-order or second-order algorithms. Compared to the implementation using only CPU (Central Processing Unit), the GPU implementation benefits from computational advantages of parallel processing for large-scale matrices and vectors operations. Numerical experiments demonstrate such computational advantages of utilizing GPU implementation in simulation optimization problems, and show that such advantage comparatively further increase as the problem scale increases.

1 INTRODUCTION

Simulation optimization (SO) generally refers to optimization in the setting where the objective function $f(x)$ and/or the constraints Θ involves uncertainty and cannot be directly analytically evaluated and can only be evaluated through simulation experiments; see Fan et al. (2024) for a recent review on simulation optimization. A general simulation optimization problem can be represented by the follows (Jian and Henderson 2015),

$$\min_{x\in\Theta} \mathbb{E}[f(x,\xi)],$$
$$\text{where } \Theta = \{x : \mathbb{E}[g(x,\xi)] \geq 0\}$$

where x are the decision variables, ξ are the random variables representing the randomness in the system, where $f(x,\xi)$ denotes one stochastic realization of objective via simulation and $g(x,\xi)$ denotes one stochastic realization of the constraint via simulation. We refer to Hong and Nelson (2009), Fu (2015), Jian and Henderson (2015), Hunter et al. (2019), Hong and Zhang (2021), Peng et al. (2023) and Fan et al. (2024) for more detailed review.

Classical implementation of simulation optimization algorithms on computers has mainly been using CPU (Central Processing Unit) by default, without a specialized use of GPU (Graphics Processing Unit). Research on parallelization and synchronization of simulation optimization algorithms has also largely been designed and implemented for CPU-based computation, or at least not specializing the use of GPU. Recent developments in the computational tools (for broad purposes) have indicated that the use of GPUs may provide specialized advantages in acceleration, if used appropriately.

In this work, we consider three sub-classes of simulation optimization problems and investigate the use of GPUs (compared to not using GPUs) to accelerate the algorithms' computational speed while maintaining a similar level of solution accuracy. Specifically, we focus on leveraging the superior capabilities of GPUs for conducting large-scale matrices and vectors operations and parallel processing to enhance the efficiency and performance of simulation optimization algorithms.

1.1 Background

The Graphics Processing Unit (GPU), originally designed for accelerating graphics rendering, has evolved into a cornerstone for parallel processing tasks. Unlike Central Processing Units (CPUs) that excel in sequential task execution with few cores, GPUs feature thousands of smaller cores optimized for handling multiple operations in parallel, making them highly efficient for parallelizable tasks (Kirk 2007; Owens et al. 2008).

In recent-year development of machine learning and deep learning, GPU with their parallel processing prowess, have significantly accelerated the training and inference processes of complex neural network architectures. This acceleration is particularly crucial for handling the vast amounts of data and the computationally intensive tasks inherent in computer vision (He et al. 2017; Gu et al. 2018). The development of transformer-based deep learning models further underscores the power of GPUs in facilitating the exploration of more advanced models. The inherent parallelism of GPUs makes them ideal for this task, enabling the rapid processing of the large-scale matrix operations that are central to transformers (Fei et al. 2017).

In our work, we attempt to study the computational prowess of GPUs within the domain of simulation optimization, which often requires continuous simulations and sampling as well as intensive matrix computations. Our work is connected to the large-scale simulation optimization literature. When the size of the SO problem gets larger, the feasible region may grow exponentially with the dimension of the decision variable. This curse of dimensionality leads to computational challenges, such as simulating exponentially more observations to estimate the objective function, low rate of convergence (Gao and Zhou 2020; Wang et al. 2023) and smoothness problem of the objective function (Ding et al. 2021; Erdogdu and Hosseinzadeh 2021). To tackle these challenges, various approaches (Kandasamy et al. 2015; Zhang et al. 2021; Rolland et al. 2018; Xu and Nelson 2013; Gao and Chen 2015; Pearce et al. 2022; Hong et al. 2022) are designed for more efficient computation and estimation. We refer to Fan et al. (2024) for more thorough review on these methods. We also refer to Eckman et al. (2023) for a broad testbed of simulation optimization problems. L'Ecuyer et al. (2017) comprehensively studied the use of GPU for random number generation. Farias et al. (2024) considered the use of GPU to accelerate policy evaluation in a general reinforcement learning problem setting. Another branch of methods in solving high-dimensional and large-scale simulation optimization problem is using parallelization (Zhang and Peng 2024; Ni et al. 2017; Luo et al. 2015). Different from the mentioned works, our work specifically focuses on the simulation optimization tasks of which the main computation can be completed through matrix operations and vectorization.

1.2 Organization and Summary

The rest of our work is organized as follows. In Section 2, we briefly overview the architecture of Graphics Processing Units (GPUs), and the mechanisms behind computation acceleration for simulation optimization with GPU implementation. In Section 3, we present the formulations of three sub-classes of optimization tasks: portfolio optimization, the multi-product Newsvendor problem, and a binary classification problem. The simulation optimization algorithms involved to address these problems include the Frank-Wolfe algorithm and the stochastic quasi-Newton algorithm.

In Section 4, we design and implement the numerical experiments via the use of GPU for the tasks and algorithms introduced in Section 3. We implement the algorithms on GPUs with JAX library and conduct a comparative analysis against their performance on CPUs across various task sizes (ranging from 100 to 1×10^6 decision variables). Our findings show that, when executed on GPUs, the algorithms operate between three to five times faster than their CPU counterparts, while maintaining similar solution accuracy and convergence. From the experiments, this running time difference between GPU version and CPU version becomes increasingly pronounced in larger-scale problems. Section 5 draws conclusions on our preliminary study of GPU implementation for some simulation optimization problems and the limitations of our work.

2 GPU FOR COMPUTATION ACCELERATION

2.1 GPU Architecture

We use the most widely used GPU Architecture: Compute Unified Device Architecture (cuda) shown in Figure 1 as example to introduce the structure of GPU and how it works for vectorization and parallel computing.

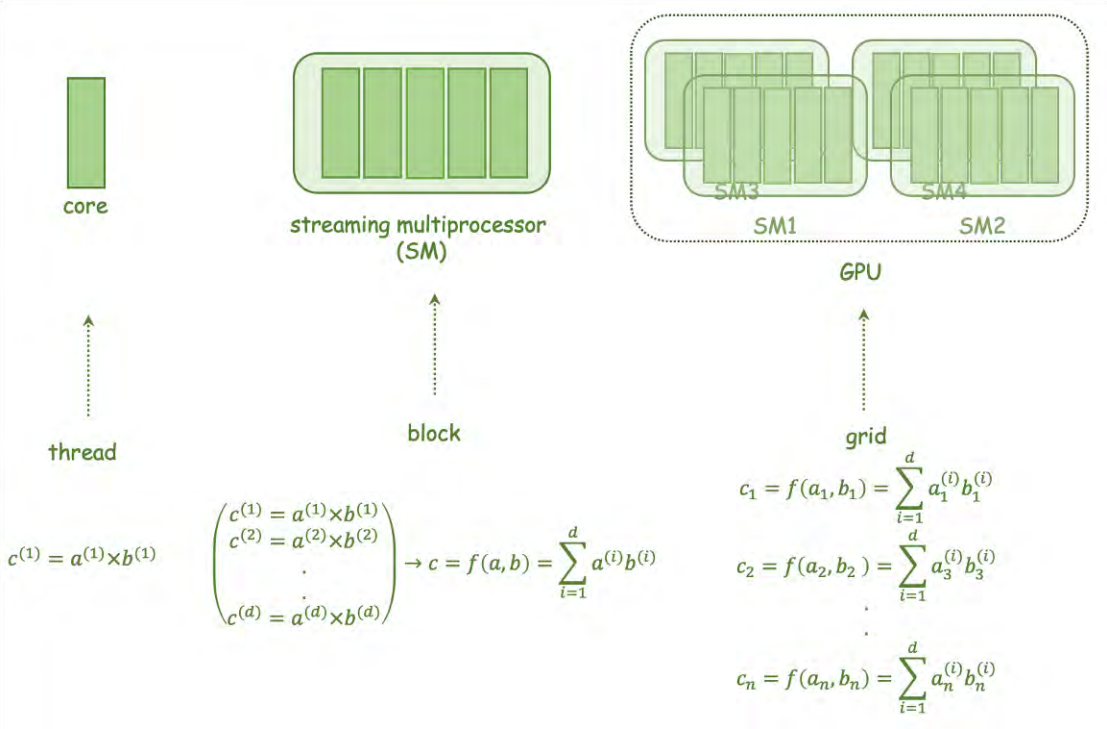

Figure 1: GPU architecture.

Notes: This figure presents the architecture of a GPU, and how each parts work parallel in a naive example for calculating inner product of two vectors. A thread refers to the calculation of $c^{(1)} = a^{(1)} \times b^{(1)}$, and is processed by a core. A block refers to the parallel operation of d threads, and is processed by a SM. A grid refers to the parallel operation of n blocks, and is processed by a GPU.

A modern GPU typically has more than thousands of cores, with each core being the most basic unit designed to carry out arithmetic operations. The arithmetic operations are referred to as 'thread' from the task perspective. The cores of a GPU are first grouped into larger units known as streaming multiprocessors (SMs), and then further organized on a grand scale to form a GPU. From the task perspective, a task processed by a SM is referred to as a 'block'; examples include vector operations. Tasks supported by the GPU, such as parallel computing of multiple functions, are referred to as 'grid'. The computational framework of GPUs follows the 'Single Instruction, Multiple Data' (SIMD) model, where multiple threads carry out the same operation but with different data sets in a block. This SIMD model naturally benefit the operations in vectorization (vector/matrix) forms.

2.2 Acceleration of Simulation Optimization with GPU

For simulation optimization tasks, especially those on a large scale, two main computational bottlenecks emerge: the first involves operations such as vector-vector, matrix-vector, and matrix-matrix multiplications, while the second pertains to the sampling process required for estimating objective values or gradients,

which demands numerous iterations, often ranging from tens to hundreds, for each computational step. The architecture of GPU is fundamentally aligned with the principles of parallel computing, making GPUs an effective platform for handling both types of operations. This parallel computing capability not only facilitates rapid execution of complex matrix and vector operations but also potentially accelerates the sampling process by enabling simultaneous execution of multiple iterations.

In the lower section of Figure 1, we briefly illustrate the operational complexities of the GPU architecture that facilitate parallel computation, exemplified by the calculation of vector inner products. Each core within the Streaming Multiprocessor (SM) executes a thread responsible for multiplying individual vector elements. As illustrated, a thread computes the product $(a_i \cdot b_i)$, with these operations occurring concurrently across all threads within a block. Within each block, inner products are computed as multiple cores work in parallel to derive the final results. Concurrently, multiple inner product functions can be executed within each grid.

Concerning the sampling process used for estimating objective values or gradients, traditional CPU-based systems usually conduct these operations sequentially, processing each sample individually. In contrast, GPUs leverage their parallel processing capabilities to handle multiple sampling operations simultaneously. Each SM in Figure 1 executes a single sampling operation for an objective function, and multiple SMs can operate in parallel to sample different pathways concurrently.

3 THREE SIMULATION OPTIMIZATION TASKS

In this section, we consider three simulation optimization tasks that all fall into the following formulation subject to deterministic linear constraints, formulated as follows:

$$\begin{aligned} \min_{\theta \in \mathbb{R}^d} f(\theta) &= \mathbb{E}\left[F(X;\theta)\right] \\ \text{s.t. } & A\theta = b. \end{aligned}$$

Many algorithms have been developed for solving this class of optimization problem. In this work, we employ the Frank-Wolfe algorithm on two specific tasks: the mean-variance portfolio optimization and the multi-product Newsvendor problem. Subsequently, we shift our focus to second-order optimization, where we implement a stochastic quasi-Newton algorithm for tackling a binary classification problem. Each of these applications is discussed in further detail below. The three algorithms under consideration predominantly depend on vectorization computations and necessitate the estimation of sample gradients. Based on the analysis in §2.2, can be accelerated through parallel computing with GPU.

3.1 Task1: Mean-variance Optimization

We first consider a general mean-variance optimization problem given by

$$\begin{aligned} \min_{\boldsymbol{w} \in \mathbb{R}^d} f(\boldsymbol{w}) &= \frac{1}{2}\mathrm{Var}[\boldsymbol{w}^\top \boldsymbol{R}] - \mathbb{E}[\boldsymbol{w}^\top \boldsymbol{R}] \\ &= \frac{1}{2}\boldsymbol{w}^\top \mathrm{Cov}[\boldsymbol{R}]\boldsymbol{w} - \boldsymbol{w}^\top \mathbb{E}[\boldsymbol{R}] \\ \text{s.t. } & \boldsymbol{w}^\top \mathbf{1} \leq 1, \\ & \boldsymbol{w} \geq 0, \end{aligned}$$

where $\boldsymbol{R}$ follows some known distribution, say $\boldsymbol{R} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Besides, we assume that $\mathbb{E}[\boldsymbol{R}]$ and $\mathrm{Cov}[\boldsymbol{R}]$ are not explicitly given, but we are allowed to draw samples from the distribution and approximate the mean and covariance matrix accordingly. Specifically, the approximated objective function is

$$\hat{f}(\boldsymbol{w}) = \frac{1}{N-1}\sum_{i=1}^{N} \boldsymbol{w}^\top (\boldsymbol{R}_i - \bar{\boldsymbol{R}})(\boldsymbol{R}_i - \bar{\boldsymbol{R}})^\top \boldsymbol{w} - \boldsymbol{w}^\top \bar{\boldsymbol{R}},$$

where

$$\bar{\boldsymbol{R}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{R}_i$$

and $\boldsymbol{R}_i, i = 1,2,\cdots,N$ are i.i.d. samples drawn from the target distribution. We apply the Frank-Wolfe algorithm to solve the problem, and we re-sample the $\boldsymbol{R}_i$'s after every M iterations. The algorithm is presented in Algorithm 1.

Algorithm 1 Frank-Wolfe Algorithm for Mean-variance Optimization

Input: Distribution of $\mathscr{D}(\boldsymbol{R})$, constraint set $W = \{\boldsymbol{w} | \boldsymbol{w}^\top \mathbf{1} \leq 1, \boldsymbol{w} \geq 0\}$, objective function f, starting point $\boldsymbol{w}_0 \in W$, number of iterations between resampling M, number of epochs K.
Output: Optimal $\boldsymbol{w}$
Initialize $\boldsymbol{w}_0$
for $k = 0$ **to** $K-1$ **do**
  Resample $\mathbf{R}_i, i = 1,\cdots,N$ from $\mathscr{D}$ ▷ Sample size can be N_k which is adapted to k
  for $m = 0$ **to** $M-1$ **do**
    Compute gradient $\nabla \hat{f}(\boldsymbol{w}_m)$
    Solve the linear subproblem $\boldsymbol{s}_m = \arg\min_{\boldsymbol{s} \in W} \boldsymbol{s}^\top \nabla \hat{f}(\boldsymbol{w}_m)$
    Compute step size $\gamma_m = \frac{2}{kM+m+2}$
    Update $\boldsymbol{w}_{m+1} = \boldsymbol{w}_m + \gamma_m(\boldsymbol{s}_m - \boldsymbol{w}_m)$
  end for
  $\boldsymbol{w}_0 \leftarrow \boldsymbol{w}_M$
end for
return $\boldsymbol{w}_0$

3.2 Task2: Multi-product Newsvendor Problem

We consider a multi-product constrained Newsvendor problem with independent product demands (Niederhoff 2007). The decision maker is interested in jointly determining the stock level x_j for product $j = 1,\cdots,N$ to satisfy overall customer demand. For each product j, the customer demand is characterized by a stochastic distribution with cdf Φ_j and pdf ϕ_j. The unit cost of product j is k_j; the holding cost per unit is h_j ($h_j < 0$ means scrap value); and the selling value per unit is v_j. Thus, the expected cost objective for product j is given by

$$f_j(x_j) = k_j x_j + h_j \int_0^{x_j} (x_j - \xi)\phi_j(\xi)\mathrm{d}\xi + v_j \int_{x_j}^{\infty} (\xi - x_j)\phi_j(\xi)\mathrm{d}\xi.$$

The stocking quantities are subject to some ex-ante linear constraints which represents the budget constraint for resources. Suppose that there are M resources to be considered, with the constraint level for resource i being C_i. The resource requirement for product j and resource i is $c_{i,j}$. For simplicity, we denote the technology matrix as $A^{M\times N} = (c_{i,j})$ and the vector of constraints as $C^{M\times 1} = (C_1,\cdots,C_M)^\top$. Non-negativity of x_j is also assumed. Therefore, the decision-making problem takes the form

$$\begin{aligned} \min_{x_1,\cdots,x_N} f(\boldsymbol{x}) &= \sum_{j=1}^{N} f_j(x_j) \\ \text{s.t. } A\boldsymbol{x} &\leq C \\ x_j &\geq 0, j = 1,\cdots,N. \end{aligned}$$

For now, we can assume that the demand for each product j follows a normal distribution $\mathscr{N}(\mu_j, \sigma_j^2)$, and the probability of negative demand is negligible as long as σ_j is small (compared with μ_j).

Since the total cost is a sum of N separable convex functions, the gradient $\nabla f(x)$ follows

$$\nabla f(\boldsymbol{x})^\top = \left(f_1'(x_1), \cdots, f_N'(x_N)\right)^\top,$$

with

$$f_j'(x_j) = k_j - v_j + (h_j + v_j)\Phi(x_j).$$

Presume that in a situation we do not have closed-form representation for $\Phi(\cdot)$, and we would approximate that in f_j' by Monte Carlo simulation. The approximated gradient is given by

$$\hat{f}_j{}'(x_j) = k_j - v_j + (h_j + v_j)\frac{1}{S_j}\sum_{j=1}^{S_j} \mathbb{I}\{d_j^{(s)} \le x_j\},$$

where $d_j^{(s)}, s = 1, \cdots, S_j$ are S_j i.i.d. samples from the demand distribution. The algorithm is provided in Algorithm 2, which is similar with Algorithm 1. For illustration purposes, we simply use Gaussian distribution in the numerical experiments.

Algorithm 2 Frank-Wolfe Algorithm for Newsvendor Problem

Input: Demand distribution for each product j, constraint set $X = \{\boldsymbol{x} | A\boldsymbol{x} \le C, \boldsymbol{x} \ge 0\}$, objective function f, starting point $x_0 \in X$, number of iterations between resampling M, number of epochs K.
Output: Optimal $\boldsymbol{x}$
Initialize $x^{(0)}$
for $k = 0$ **to** $K-1$ **do**
    For each j, resample $d_j^{(s)}, s = 1, \cdots, S_j$ from $\mathcal{N}(\mu_j, \sigma_j^2)$
    for $m = 0$ **to** $M-1$ **do**
        Compute gradient $\nabla \hat{f}(\boldsymbol{x}^{(m)}) = \left(\hat{f}_1(x_1^{(m)}), \cdots, \hat{f}_N(x_N^{(m)})\right)^\top$ according to (3.2)
        Solve the linear subproblem $\boldsymbol{s}^{(m)} = \arg\min_{\boldsymbol{s} \in X} \boldsymbol{s}^\top \nabla \hat{f}(\boldsymbol{x}^{(m)})$
        Compute step size $\gamma_m = \frac{2}{kM+m+2}$
        Update $\boldsymbol{x}^{(m+1)} = \boldsymbol{x}^{(m)} + \gamma_m(\boldsymbol{s}^{(m)} - \boldsymbol{x}^{(m)})$
    end for
    $x^{(0)} \leftarrow x^{(M)}$
end for
return $x^{(0)}$

3.3 Task3: Binary Classification Problem

In this section, we consider second-order quasi-Newton-type algorithms. Specifically, we consider a binary classification problem from Byrd et al. (2016), given as

$$\min_{\omega} F(\omega) = -\frac{1}{N}\sum_{i=1}^{N} z_i \log(c(\omega; x_i)) + (1 - z_i)\log(1 - c(\omega; x_i)),$$

where

$$c(\omega, x_i) = \frac{1}{1 + \exp(-x_i^T \omega)},\ x_i \in \mathbb{R}^n,\ \omega \in \mathbb{R}^n,$$

and $z_i \in \{0, 1\}$. To explain, x_i $(i = 1, 2, \ldots, N)$ denote the feature values of each data point i, and z_i is the corresponding classification label. The n data points may come from sample collection or Monte Carlo

simulation, depending on different application cases. The objective function is minimized (w.r.t. ω) to derive a function $c_{\omega^*}(x_i)$ that best predicts the label z_i of the sample x_i.

To solve the problem, we apply the GPU implementation of the stochastic quasi-Newton method (SQN) provided in Byrd et al. (2016) as given in Algorithm 3. Specifically, we use a mini-batch stochastic gradient based on $b = |\mathscr{S}|$ sampled pairs of (x_i, z_i), yielding the following estimate of gradient

$$\hat{\nabla} F(\omega) = \frac{1}{b} \sum_{i \in \mathscr{S}} \nabla f(\omega; x_i, z_i),$$

where $f(\omega; x_i, z_i) = z_i \log(c(\omega; x_i)) + (1 - z_i) \log(1 - c(\omega; x_i))$. Further, let

$$\hat{\nabla}^2 F(\omega) = \frac{1}{b_H} \sum_{i \in \mathscr{S}_H} \nabla^2 f(\omega; x_i, z_i) \tag{1}$$

be a sub-sampled Hessian, where $\mathscr{S}_H \in \{1, \ldots, N\}$ is also randomly sampled, and $b_H = |\mathscr{S}_H|$. The rest of the details are given in Algorithm 3.

Algorithm 3 SQN Algorithm for the Classification Problem

Input: Step parameter integer $L > 0$, memory integer $M > 0$, step length parameter $\beta > 0$, sample size parameters b, b_H; initial point ω^1
Output: Optimal ω
Set $t = -1$, $\bar{\omega}_t = 0$. ▷ t records number of correction pairs currently computed
for $k = 1, 2, \ldots$ **do**
  Choose a sample set $\mathscr{S} \in \{1, 2, \ldots, N\}$
  Calculate stochastic gradient $\hat{\nabla} F(\omega_t)$ by (3.3).
  $\bar{\omega}_t = \bar{\omega}_t + \omega^k$, $\alpha_k = \beta / k$
  if $k \leq 2L$ **then**
    $\omega^{k+1} = \omega^k - \alpha^k \hat{\nabla} F(\omega^k)$ ▷ Stochastic gradient iteration
  else
    $\omega^{k+1} = \omega^k - \alpha^k H_t \hat{\nabla} F(\omega^k)$, where H_t is calculated by Algorithm 4.
  end if
  if mod $(k, L) = 0$ **then** ▷ Compute correction pairs every L iterations
    $t = t + 1$
    $\bar{\omega}_t = \bar{\omega}_t / L$.
    if $t > 0$ **then**
      Choose a sample $\mathscr{S}_H \in \{1, \cdots, N\}$ to compute $\hat{\nabla}^2 F(\bar{\omega}_t)$ by (1).
      Compute $s_t = (\bar{\omega}_t - \bar{\omega}_{t-1})$, $y_t = \hat{\nabla}^2 F(\bar{\omega}_t)(\bar{\omega}_t - \bar{\omega}_{t-1})$. ▷ Correction pairs
    end if
    $\bar{\omega}_t = 0$
  end if
end for
return ω^k

4 NUMERICAL RESULTS

In this section, we study the numerical performance of the designed algorithm implemented on GPU for three mentioned simulation optimization tasks. In particular, we compare the implemented simulation optimization algorithms on GPU with their counterpart versions on CPU for the computation time and accuracy.

Algorithm 4 Hessian Updating

Input: Updating counter t, memory integer $M > 0$, and correction pairs (s_j, y_j) where $j = t-\tilde{m}+1,\ldots,t$ and $\tilde{m} = \min\{t, M\}$ ▷ All come from Algorithm 3

Output: new matrix H_t

set $H = (s_t^T y_t)/(y_t^T y_t)I$, where s_t and y_t are computed from Algorithm 3.

for $j = t-\tilde{m}+1,\ldots,t$ **do**

$\rho_j = 1/y_j^T s_j$

Apply BFGS formula:

$$H \leftarrow (I - \rho_j s_j y_j^T) H (I - \rho_j y_j s_j^T) + \rho_j s_j s_j^T$$

end for

return $H_t \leftarrow H$

4.1 Experimental Setup

We use JAX (Bradbury et al. 2018) library for implementation. We executed the same algorithm for each task using identical parameters on both CPU and GPU (with JAX) across problems of varying sizes to illustrate the performance discrepancies between CPU and GPU at different scales. Our primary focus was on the computational time metric, which we estimated based on the computation time required for comparable iterations of the algorithms on both platforms. Additionally, we evaluated the accuracy and convergence metrics, defined by the relative squared error (RSE) in each iteration relative to the final objective values. The parameters of each task are the same for GPU and CPU settings, which are listed below.

For Task 1, we consider the mean-variance optimization problem of asset sizes of $5\times10^2, 5\times10^3, 1\times10^4, 5\times10^4, 1\times10^5$. For each task, we run $K = 1500$ iterations of estimation and for each time of estimation of gradient, we sample $M = 25$ times for all asset sizes except for 1×10^5 where we sample $M = 50$ times for better estimation in extra high-dimensional cases. The μ_i are randomly generated from Uniform$(-1,1)$ and σ_i are randomly generated from Uniform$(0, 0.025)$.

For Task 2, we consider the news-vendor problem of inventory sizes of $1\times10^2, 1\times10^3, 1\times10^4, 1\times10^5$, and 1×10^6. For each task, we run $K = 1500$ iterations of estimation and for each time of estimation of gradient, we sample $M = 25$ times for all asset sizes except for 1×10^6 where we sample $M = 50$ times. The μ_i are randomly generated from Uniform$(20, 50)$ and σ_i are randomly generated from Uniform$(10, 20)$.

For Task 3, we use the synthetic data method from Mukherjee et al. (2013) and Byrd et al. (2016). The synthetic dataset contain N sample each has n binary features. Here the feature size n is the size of the problem. We generate datasets with $50, 500, 1000, 5000$ features, each with sample size $N = 30n$. The labels are generated by a random linear combination of the features, and contain 10% white label noise for binary classification. The stepsize is set as $\alpha_k = \beta/k$. Other parameters are given as $M = 25$, $L = 10$, $b = 50$, $\beta = 2$ and $b_H = 300$ or 600. For each round of experiment we run $K = 2000$ iterations.

All our experiments were conducted in a Python 3 environment. For CPU computations, we utilized an AMD Ryzen Threadripper 3970X with 256GB of memory, and for GPU computations, we employed an NVIDIA GeForce RTX 3090 with 24GB of memory, with key parameters are detailed in Table 1. The CPU and GPU used in the experiment are at comparable market price.

Table 1: Comparison of CPU and GPU specifications.

	CPU	**GPU**
Processor	AMD Ryzen Threadripper 3970X	NVIDIA GeForce RTX 3090
Theoretical peak (FP32)	108 GFLOPS	35.58 TFLOPS
Maximum memory bandwidth	172.73 GB/sec	936.2 GB/sec

4.2 Experiment Results

In Figure 2, we present the average computation time and the corresponding confidence intervals, defined as plus or minus two standard deviations, for three tasks of varying scales. Additionally, we examine the convergence properties of each task using a selected example size, demonstrating the relative squared error (RSE) of the objective values in comparison to the final objective values.

Figure 2 illustrates that the GPU implementation consistently outperforms in computational time across all three SO tasks. As the problem size increases, the benefits of leveraging GPU implementation for parallel computing and vectorization become increasingly pronounced. For instance, in portfolio optimization tasks involving 10^5 assets, completing all iterations typically requires around 6 hours. Using GPU technology, however, can reduce this iteration time to approximately 1 hour, thus achieving an acceleration factor of about six. Additionally, Table 2 demonstrates that the same algorithm running on both GPU and CPU achieves nearly identical levels of accuracy at various iteration steps. This similarity in performance is anticipated since, apart from the computation hardware, all other parameters remain the same throughout the process. Additionally, from an energy consumption perspective, the average power consumption rate of the CPU during the experiments is about 220W and the GPU's power consumption is about 320W. Therefore, considering the approximately sixfold acceleration in experiment time, the use of GPU implementation achieves about 75% energy savings compared to CPU implementation to achieve the same precision goal.

Table 2: Evaluation of performance on different tasks with adjusted error estimates.

	Asset (5k)		Inventory (10k)		Classification (1k)	
	GPU	CPU	GPU	CPU	GPU	CPU
RSE at iteration 50	85.07%	83.19%	89.92%	88.73%	72.16%	76.25%
	(± 9.74%)	(± 10.65%)	(± 7.02%)	(± 7.33%)	(± 8.44%)	(± 7.74%)
RSE at iteration 100	62.41%	63.71%	76.25%	72.93%	51.06%	53.46%
	(± 5.46%)	(± 4.86%)	(± 8.49%)	(± 9.45%)	(± 5.92%)	(± 5.10%)
RSE at iteration 500	24.07%	25.62%	40.94%	38.52%	31.29%	29.67%
	(± 4.97%)	(± 5.87%)	(± 8.11%)	(± 8.53%)	(± 4.07%)	(± 5.21%)
RSE at iteration 1000	13.39%	12.93%	20.58%	23.67%	15.59%	16.77%
	(± 2.86%)	(± 3.96%)	(± 5.78%)	(± 6.48%)	(± 4.00%)	(± 3.71%)

Notes: We define the Relative Squared Error (RSE) as $\text{RSE} = \left(\frac{y^{(t)}-y^*}{y^{(t)}}\right)^2 \times 100\%$, where y^* represents the final objective value upon completion of iterations, and $y^{(t)}$ denotes the objective value at the t^{th} iteration. This table presents the RSE for various optimization tasks: a mean-variance optimization involving 5000 assets, a newsvendor problem with 10,000 products, and binary classification tasks with 1000 features, each assessed under varying iteration step counts within a total of 10,000 iteration steps. We repeat each experiment for 7 repetitions.

5 CONCLUSION

In this paper, we present a preliminary study on employing GPUs to expedite computation across three simulation optimization tasks. We observe that by leveraging the GPU's capabilities for fast vectorization and parallel computing, both first-order and second-order algorithms experience a performance improvement of approximately 3 to 6 times. The relative benefit increases as the problem scale increases. Our study has limitations, including reliance on third-party GPU acceleration packages, which may not fully utilize the computational power of GPUs. Additionally, we have not thoroughly investigated the specific contributions of GPUs at various computational stages. Moreover, our focus is restricted to gradient-based methods, and has not extended to other simulation optimization algorithms. We also note that this work does not dive into the different ways of generation of random samples on GPUs, which is of independent but important interest.

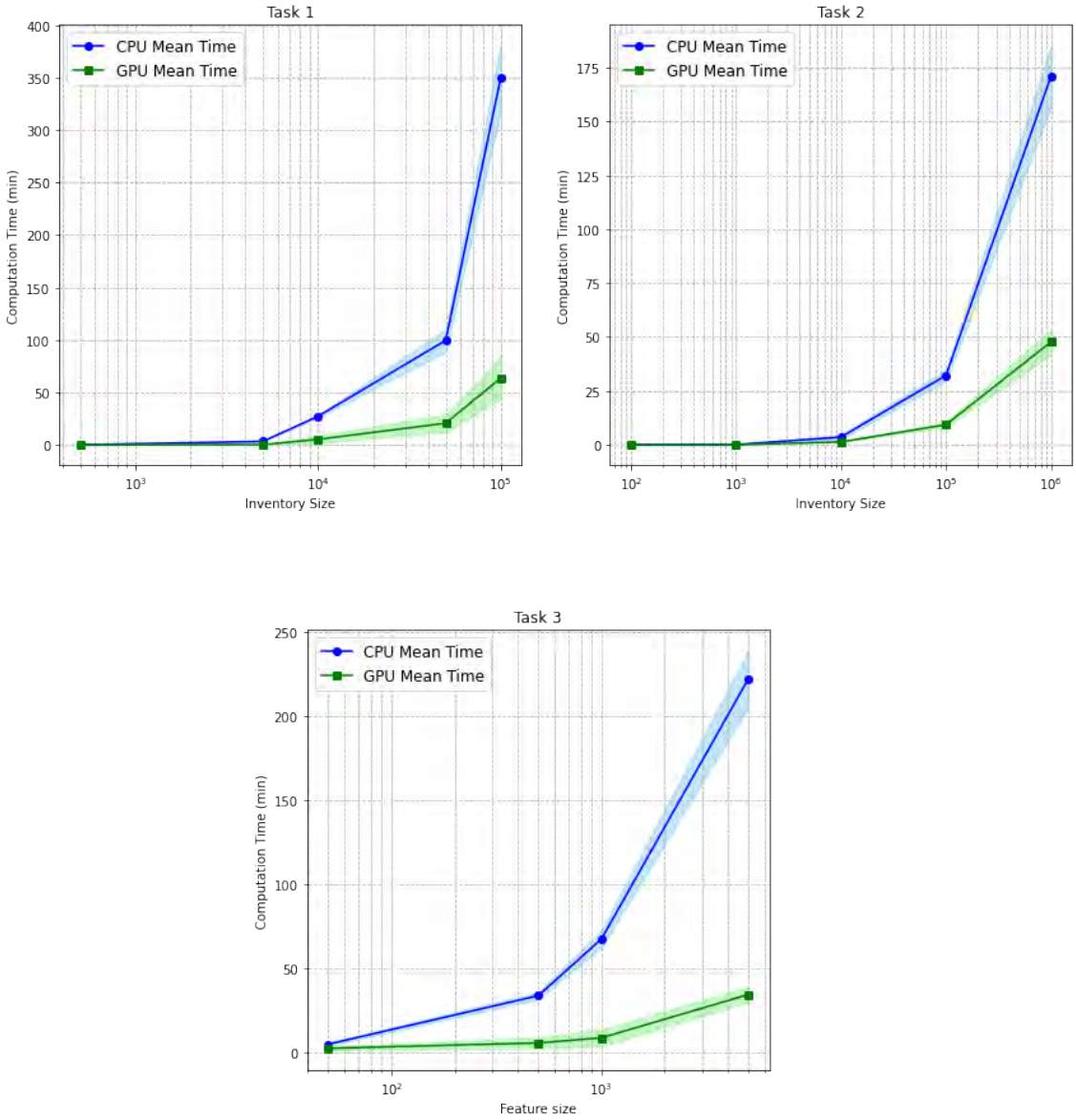

Figure 2: Computation time for three tasks with different size.

Notes: This figure demonstrates the computation time and corresponding $\pm 2\sigma$ confidence interval for three considered tasks of different sizes.

ACKNOWLEDGMENTS

We deeply thank the three anonymous reviewers for their comments and suggestions. We find all of them very helpful to improve our manuscript.

REFERENCES

Bradbury, J., R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin *et al.* 2018. "JAX: Composable Transformations of Python+ NumPy Programs".

Byrd, R. H., S. L. Hansen, J. Nocedal, and Y. Singer. 2016. "A Stochastic Quasi-Newton Method for Large-Scale Optimization". *SIAM Journal on Optimization* 26(2):1008–1031.

Ding, L., R. Tuo, and X. Zhang. 2021. "High-Dimensional Simulation Optimization via Brownian Fields and Sparse Grids". *arXiv Preprint arXiv:2107.08595*.

Eckman, D. J., S. G. Henderson, and S. Shashaani. 2023. "SimOpt: A Testbed for Simulation-Optimization Experiments". *INFORMS Journal on Computing* 35(2):495–508.

Erdogdu, M. A. and R. Hosseinzadeh. 2021. "On the Convergence of Langevin Monte Carlo: The Interplay Between Tail Growth and Smoothness". In *Conference on Learning Theory*, 1776–1822. PMLR.

Fan, W., L. J. Hong, G. Jiang, and J. Luo. 2024. "Review of Large-Scale Simulation Optimization". *arXiv Preprint arXiv:2403.15669*.

Farias, V., J. Gijsbrechts, A. Khojandi, T. Peng and A. Zheng. 2024. "Massive Speedups for Policy Evaluation in Inventory Management". *Working Paper*.

Fei, C., F. C. Lee, and Q. Li. 2017. "High-Efficiency H igh-Power-Density L LC C onverter w ith a n I ntegrated P lanar Matrix Transformer for High-Output Current Applications". *IEEE Transactions on Industrial Electronics* 64(11):9072–9082.

Fu, M. C. 2015. *Handbook of Simulation Optimization*, Volume 216. Springer.

Gao, S. and W. Chen. 2015. "A Note on the Subset Selection for Simulation Optimization". In *2015 Winter Simulation Conference (WSC)*, 3768–3776 https://doi.org/10.1109/WSC.2015.7408534.

Gao, W. and Z.-H. Zhou. 2020. "Towards Convergence Rate Analysis of Random Forests for Classification". *A dvances in Neural Information Processing Systems* 33:9300–9311.

Gu, J., Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai *et al*. 2018. "Recent Advances in Convolutional Neural Networks". *Pattern Recognition* 77:354–377.

He, K., G. Gkioxari, P. Dollár, and R. Girshick. 2017. "Mask R-CNN". In *Proceedings of the IEEE International Conference on Computer Vision*, 2961–2969.

Hong, L. J., G. Jiang, and Y. Zhong. 2022. "Solving Large-Scale Fixed-Budget Ranking and Selection Problems". *INFORMS Journal on Computing* 34(6):2930–2949.

Hong, L. J. and B. L. Nelson. 2009. "A Brief Introduction to Optimization via Simulation". In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, 75–85 https://doi.org/10.1109/WSC.2009.5429321.

Hong, L. J. and X. Zhang. 2021. "Surrogate-Based Simulation Optimization". In *Tutorials in Operations Research: Emerging Optimization Methods and Modeling Techniques with Applications*, 287–311. INFORMS.

Hunter, S. R., E. A. Applegate, V. Arora, B. Chong, K. Cooper, O. Rincón-Guevara *et al*. 2019. "An Introduction to Multiobjective Simulation Optimization". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 29(1):1–36.

Jian, N. and S. G. Henderson. 2015. "An Introduction to Simulation Optimization". In *2015 Winter Simulation Conference (WSC)*, 1780–1794 https://doi.org/10.1109/WSC.2015.7408295.

Kandasamy, K., J. Schneider, and B. Póczos. 2015. "High-Dimensional Bayesian Optimisation and Bandits via Additive Models". In *International Conference on Machine Learning*, 295–304. PMLR.

Kirk, D. 2007. "NVIDIA CUDA Software and GPU Parallel Computing Architecture". In *International Symposium on Memory Management*, Volume 7, 103–104.

Luo, J., L. J. Hong, B. L. Nelson, and Y. Wu. 2015. "Fully Sequential Procedures for Large-Scale Ranking-and-Selection Problems in Parallel Computing Environments". *Operations Research* 63(5):1177–1194.

L'Ecuyer, P., D. Munger, B. Oreshkin, and R. Simard. 2017. "Random Numbers for Parallel Computers: Requirements and Methods, with Emphasis on GPUs". *Mathematics and Computers in Simulation* 135:3–17.

Mukherjee, I., K. Canini, R. Frongillo, and Y. Singer. 2013. "Parallel Boosting with Momentum". In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III*, 17–32. Springer.

Ni, E. C., D. F. Ciocan, S. G. Henderson, and S. R. Hunter. 2017. "Efficient R anking a nd S election i n P arallel Computing Environments". *Operations Research* 65(3):821–836.

Niederhoff, J. A. 2007. "Using Separable Programming to Solve the Multi-Product Multiple Ex-Ante Constraint Newsvendor Problem and Extensions". *European Journal of Operational Research* 176(2):941–955 https://doi.org/https://doi.org/10.1016/j.ejor.2005.09.046.

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.

Pearce, M. A. L., M. Poloczek, and J. Branke. 2022. "Bayesian Optimization Allowing for Common Random Numbers". *Operations Research* 70(6):3457–3472.

Peng, Y., C.-H. Chen, and M. C. Fu. 2023. "Simulation Optimization in the New Era of AI". In *Tutorials in Operations Research: Advancing the Frontiers of OR/MS: From Methodologies to Applications*, 82–108. INFORMS.

Rolland, P., J. Scarlett, I. Bogunovic, and V. Cevher. 2018. "High-Dimensional Bayesian Optimization via Additive Models with Overlapping Groups". In *International Conference on Artificial I ntelligence a nd S tatistics*, 2 98–307. PMLR.

Wang, X., L. J. Hong, Z. Jiang, and H. Shen. 2023. "Gaussian Process-Based Random Search for Continuous Optimization via Simulation". *Operations Research*.

Xu, W. L. and B. L. Nelson. 2013. "Empirical Stochastic Branch-and-Bound for Optimization via Simulation". *IIE Transactions* 45(7):685–698.

Zhang, S., F. Yang, C. Yan, D. Zhou and X. Zeng. 2021. "An Efficient Batch-Constrained Bayesian Optimization Approach for Analog Circuit Synthesis via Multiobjective Acquisition Ensemble". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41(1):1–14.

Zhang, Z. and Y. Peng. 2024. "Sample-Efficient Clustering and Conquer Procedures for Parallel Large-Scale Ranking and Selection". *arXiv Preprint arXiv:2402.02196*.

AUTHOR BIOGRAPHIES

JINGHAI HE is a second-year Ph.D. student in the Department of Industrial Engineering & Operations Research at the University of California Berkeley. His research interests lie in reinforcement learning, generative models and sequential decision-making. His email address is jinghai_he@berkeley.edu.

HAOYU LIU is a first-year Ph.D. student in the Department of Industrial Engineering & Operations Research at the University of California Berkeley. His research interests lie in simulation, A/B testing and generative AI. His email address is haoyuliu@berkeley.edu.

YUHANG WU is a third-year Ph.D. student in the Department of Industrial Engineering & Operations Research at the University of California Berkeley. His email address is wuyh@berkeley.edu.

ZEYU ZHENG is an associate professor in the Department of Industrial Engineering & Operations Research at the University of California Berkeley. His email address is zyzheng@berkeley.edu.

TINGYU ZHU is a second-year Ph.D. student in the Department of Industrial Engineering & Operations Research at the University of California Berkeley. She has done research in simulation, sequential experiments and generative AI. Her email address is tingyu_zhu@berkeley.edu.