risks

MDPI

Article

Distributed Least-Squares Monte Carlo for American Option Pricing

Lu Xiong *, Jiyao Luo, Hanna Vise and Madison White

Department of Mathematical Sciences, College of Basic and Applied Sciences, Middle Tennessee State University, Murfreesboro, TN 37132, USA; jl2an@mtmail.mtsu.edu (J.L.); hv2s@mtmail.mtsu.edu (H.V.); mbw5k@mtmail.mtsu.edu (M.W.)

* Correspondence: lu.xiong@mtsu.edu

Abstract: Option pricing is an important research field in financial markets, and the American option is a common financial derivative. Fast and accurate pricing solutions are critical to the stability and development of the market. Computational techniques, especially the least squares Monte Carlo (LSMC) method, have been broadly used in optimizing the pricing algorithm. This paper discusses the application of distributed computing technology to enhance the LSMC in American option pricing. Although parallel computing has been used to improve the LSMC method, this paper is the first to explore distributed computing technology for LSMC enhancement. Compared with parallel computing, distributed computing has several advantages, including reducing the computational complexity by the "divide and conquer" method, avoiding the complicated matrix transformation, and improving data privacy as well as security. Moreover, LSMC is suitable for distributed computing because the price paths can be simulated and regressed separately. This research aims to show how distributed computing, particularly the divide and conquer approach implemented by Apache Spark, can be used to improve the efficiency and accuracy of LSMC in American option pricing. This paper provides an innovative solution to the financial market and could contribute to the advancement of American option pricing research.

Keywords: American option pricing; least squares Monte Carlo (LSMC); distributed computing; computational complexity; MapReduce; Apache Spark

Citation: Xiong, Lu, Jiyao Luo, Hanna Vise, and Madison White. 2023. Distributed Least-Squares Monte Carlo for American Option Pricing. *Risks* 11: 145. https://doi.org/10.3390/risks11080145

Academic Editors: Tianyang Wang, Jing Ai and Xiufang Li

Received: 13 July 2023
Revised: 3 August 2023
Accepted: 4 August 2023
Published: 8 August 2023

1. Introduction and Literature Review

1.1. Introduction

Option pricing is an important research field within financial markets and has attracted a considerable amount of attention from both academics and practitioners for a long time. The American option, a common financial derivative, requires accurate and quick pricing solutions, which are critical for the stability and development of the financial market. With the significant advancements in computational technologies, computerized computational methods have been widely used for optimal pricing. Among these, least squares Monte Carlo (LSMC), a relatively new and effective technology, has been utilized to solve the American option pricing problem. In this paper, we discuss the application of distributed computing technology to enhance the LSMC method in terms of American option pricing.

Currently, there are several researchers (Weigel 2018) who have used the method of parallel computing to improve the LSMC method, but, for the first time, we use this paper to discuss distributed computing technology to improve the LSMC method. In parallel computing, the computer's memory is shared, but, in distributed computing, the memory is separated so that it can be referenced as a single shared space that runs independently. The method of distributed computing has several advantages over parallel computing. Firstly, distributed computing can reduce the overall computational complexity of the algorithm by using the "divide and conquer" method for the total computational complexity. The sum

from each computer node is less than the original total computational complexity executed on the single note without using distributed computing. This advantage was supported by the mathematical theory of distributed regression (Dobriban and Sheng 2021). Secondly, compared with the traditional parallel least squares regression, distributed regression does not require complicated matrix transformation techniques. Thirdly, distributed computing can help protect the privacy and security of the given data. These data can be stored in the original locations of the distributed system, and moving the data around will not be required. Lastly, the LSMC method itself is suitable to be computed distributively because the paths can be simulated and regressed separately.

The goal of this paper is to show through research how distributed computing, especially distributed regression technology, can be used to improve the LSMC method regarding the efficiency and accuracy of American option pricing. Our paper provides a new solution to the financial market, and we expect it can provide innovation and progress in the field of American option pricing.

The rest of the paper is organized as follows. In Section 2, we discuss the generic Monte Carlo simulation for American option pricing and its computational complexity. Comparatively, Section 3 introduces the least squares Monte Carlo simulation in which we discuss its advantages and limitations for option pricing. In Section 4, we discuss distributed regression to the LSMC method to further speed up its computational process. Section 5 provides detailed computational complexity comparisons of other option pricing approaches in which we rank which ones are the most accurate and fast. Next, Section 6 aims to further validate the distributed LSMC by performing an experiment using PySpark 3.3.1 and Python 3.9.7. Finally, we conclude the paper in Section 7.

1.2. Literature Review

The Black–Scholes model was developed by Black and Scholes (1973) to calculate the price of European options using a differential equation with various variables, such as the current stock price, strike price, risk-free rate, volatility, and time to maturity or expiration. Although the Black–Scholes model can provide an analytical solution for European options, it is not applicable to American options due to their added complexity as they can be exercised before the expiration date. For American option pricing, numerical methods such as Monte Carlo simulation and the binomial tree model are typically used to estimate the option value.

The binomial tree model, originally introduced by Cox et al. (1979), was developed as a visual illustration of the Black-Scholes model by the three financial educators. The binomial tree model relies upon the same variables as the Black–Scholes model to create an array of pathways to view specific stages of an option price. The option price will fluctuate up or down depending on the interest or discounted rate, length of time, and volatility that the option contract is subject to. With the increase in the time steps, the number of nodes in the binomial tree increases exponentially, which makes the computational complexity significantly higher (Dai and Lyuu 2010). The computing needs for a large number of nodes could result in too much time in computing, especially when high-accuracy pricing is required.

Tilley (1993) first proposed using the Monte Carlo method for American option pricing, but this solution has not seen widespread use due to issues such as its complexity. The breakthrough in this area came with Longstaff and Schwartz (2001), who introduced the least squares Monte Carlo (LSMC) method to ascertain the optimal relationship between the continuous value of the derivative security and the value of the relevant variable at each moment. They then determined whether the option should be executed in advance at a specific moment. Currently, the LSMC method has become the standard method for pricing American options. This method estimates the option price by generating various paths that allow for the observation of multiple price points and the determination of the optimal point at which to exercise the option. The paths are determined by a range of variables, such as stock price, option price, time to maturity, risk-free rate, frequency,

dividends, and volatility. LSMC has many applications; for instance, it has become a popular proxy technique for reducing the computational complexity of Solvency Capital Requirement (SCR) under the Solvency II framework (Bauer et al. 2010; Krah et al. 2018). Proxy techniques are mathematical approximation methods used to estimate a quantity of interest when it is too difficult to measure directly. Under fairly general conditions, Clément et al. (2002) proved the almost sure convergence of the LSMC algorithm. They also determined the rate of convergence of approximation and proved that its normalized error is asymptotically Gaussian. Haugh and Kogan (2004) developed a dual method Monte Carlo for pricing American options. This method constructs upper and lower bounds on the true price of the option using any approximation to the option price. Numerical results suggest that this approach can be successfully applied to problems of practical interest.

In the field of pricing American options using machine learning, Goudenège et al. (2019) proposed an efficient method to compute the price of multi-asset American options based on machine learning, Monte Carlo simulations, and the variance reduction technique. Chen and Wan (2021) proposed a deep neural network framework for computing prices and deltas of American options in high dimensions.

Datta (1985) proposed an effective parallel algorithm that uses matrix decomposition techniques such as QR decomposition or Cholesky decomposition. Chen et al. (2015) accelerated the least squares Monte Carlo method with parallel computing. Zhang et al. (2015) proposed a "divide and conquer" type of distributed regression algorithm. Dobriban and Sheng (2021) provide a mathematical theory of distributed regression. However, there is no research studying the application of distributed computing in LSMC, an area this paper intends to explore.

2. Generic Monte Carlo for American Option Pricing

Consider an American put option, assuming that T is the expiration time, where T^* is the optimal exercise time, and S_t is the asset price at time t. The price of the American put option at time 0 is given by

$$f = e^{-rT^*} E^Q[f(S_0, S_1, \cdots, S_{T^*}, \cdots, S_T)] \tag{1}$$

where $E^Q[f(S_0, S_1, \cdots, S_{T^*}, \cdots, S_T)]$ is the risk neutral measure.

To perform a Monte Carlo simulation, we assume that the stochastic process S_t, which is the asset price, follows geometric Brownian motion, so it satisfies the following stochastic differential equation (SDE):

$$dS = Srdt + S\sigma dz \tag{2}$$

where r is the drift or risk-free interest rate of the asset, σ is the volatility of the asset, and z is a Brownian motion. According to Itô's formula, (2) derives

$$d\ln S = \left(r - \frac{\sigma^2}{2}\right)dt + \sigma dz \tag{3}$$

Using the discretization method, we divide the whole time interval $[0, T]$ to N sub-intervals: $\Delta t = T/N$, and then we can derive the following approximate recurrence formula from (3):

$$\ln(S_i) - \ln(S_{i-1}) = \left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t} \cdot Z_i \tag{4}$$

where ϵ_i follows the standard normal distribution. From (4), we can further derive the following formula for S_i:

$$S_i = \exp\left(\ln S_0 + i \cdot \left[\left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t} \cdot Z_i\right]\right) \tag{5}$$

After simulating M paths, denoted as $h_j(S_0, S_1, \cdots, S_T),\ j \in \{1, 2, \cdots, M\}$, we obtain an $M \times (N+1)$ matrix representing the price paths.

The intrinsic value or the early exercise payoff for an American put option of path j at time i is $I_i^j\left(S_i^j\right) = \max\{K - S_i^j, 0\}$ when the asset price is S_i^j. The option price of path j at time i is

$$f_i^j(S_i^j) = \max\left\{ I_i^j\left(S_i^j\right), E^Q\left[e^{-r\Delta t} f_{i+1}^j\left(S_{i+1}^j\right)\middle| S_i^j\right]\right\} \tag{6}$$

Computing the continuation value $E^Q\left[e^{-r\Delta t} f_{i+1}^j\left(S_{i+1}^j\right)\middle| S_i^j\right]$ in Formula (6) of the generic Monte Carlo requires a full revaluation of the option price at each time step with a new set of random paths. This essentially represents a "nested Monte Carlo" simulation, which is extremely time- and memory-consuming.

The computational complexity of the generic Monte Carlo simulation for formula (6) is $O(C^N)$, where $C \geq 2$ is a positive integer and N is the number of sub-intervals.

3. LSMC

3.1. Introduction of LSMC

The LSMC method originates from non-parametric regression. The goal of non-parametric regression in statistics is to estimate the conditional expectations: $E(S_t \mid C_t) = f(S_t)$, where S_t and C_t are the response variable (stock price) and covariate (continuation value), respectively. In statistical problems, observations of S_t and C_t are accessible, and non-parametric regression methods can estimate the function $f(S_t)$ based on the samples. It is called non-parametric regression because the form of the function $f(S_t)$ requires minimal assumptions and can be a general non-linear function. Geometric Brownian motion can also be used to simulate future stock prices. The LSMC simulation is similar to a traditional model in that it involves the allocation of the remaining time of an option's expiration. A random sample of the asset price path is generated, and the expected return of the option is compared with the expected return of the underlying asset. If the exercise value exceeds the expected holding value, it is the ideal strategy to sell the option immediately. Least squares regression is used for the approximation of the continuation values in which each continuation value is a function of the stock price. We can denote the continuation value as follows:

$$C_t \approx a_2 \cdot S_t^2 + a_1 \cdot S_t + b \tag{7}$$

The generic (nested) Monte Carlo method of option pricing applies the payoff function and takes the discounted expectation of those paths' payouts to obtain the option price. However, computing continuation values at future time points becomes less straightforward with this approach. The generic Monte Carlo method tends to be inefficient when dealing with large sets of constraints. One could complete this with nested simulations, but this would not be practical since the computational complexity $O(C^N)$, where C is the number of simulated scenarios at each time point), is so large that we need to introduce the LSMC algorithm to reduce the running time of the generic MC.

To improve the efficiency of the nested Monte Carlo simulation, the LSMC method was first proposed by Longstaff and Schwartz (2001). This method involves simulating the paths and then working backward in time to estimate the continuation values through least squares regression. The fitted regression curve can be used to average the errors, producing a more accurate solution through a faster process.

3.2. Algorithm Steps of LSMC

Step 1 Initialization and simulation of asset price paths.

Initialize the parameters of the problem, such as the initial underlying asset price $S(0)$, volatility σ, risk-free interest rate r, time to maturity T, number of time steps N, and the number of simulated patterns M. Use geometric Brownian motion to simulate the M price paths of the asset from time $t = 0$ to $t = T$ with N time steps. Given the asset price at the time step $i - 1$, the asset price at the ith time step can be derived using the following equation:

$$S_i^j = S_{i-1}^j e^{\left(\left(r-\frac{1}{2}\sigma^2\right)\Delta t+\sigma\sqrt{\Delta t}\cdot Z_i^j\right)} \tag{8}$$

where Z_i^j is a standard normal random variable, and j is the path index.

Step 2 Valuatiton of the option at maturity

Calculate the option payoff at maturity for each simulated path. For an American put option, that payoff is

$$f_N^j\left(S_N^j\right) = \max\left(K - S_N^j, 0\right), \tag{9}$$

where K is the strike price.

Step 3 Backward induction.

Starting from time step $i = N - 1$ and going backward to $i = 0$, perform the following steps:

1. Regression: Conduct the least square regression for the continuation value at time $i + 1$ using the basis functions, which take the underlying asset price at time i as input. The regression model is

$$Y_{i+1}^j = a_0 + a_1 S_i^j + a_0\left(S_i^j\right)^2 + \varepsilon_{i+1}^j \tag{10}$$

where $Y_{i+1}^j = e^{-r\Delta t} f_{i+1}^j\left(S_{i+1}^j\right)$ is the discounted continuation value of the option at time $i + 1$.

2. Decide on continuation value or early exercise: Compute the continuation value $C_i^j = a_0 + a_1 S_i^j + a_2 (S_i^j)^2$ and compare it with the early exercise payoff. If the latter is larger, then set $f_i^j\left(S_i^j\right) = \max(K - S_i^j, 0)$; otherwise, set $f_i^j\left(S_i^j\right) = C_i^j$.

Step 4 Estimation of the option price.

Once we reach time $i = 0$, the estimated option price is the average of the discounted option value $f_0^j\left(S_0^j\right) = e^{-r\Delta t} f_1^j\left(S_1^j\right)$ over the M paths (Longstaff and Schwartz 2001):

$$V_0 = \frac{1}{M}\sum_{j=1}^{M} f_0^j\left(S_0^j\right) \tag{11}$$

Figure 1 shows the algorithm flow chart of LSMC.

3.3. Convergence and Computational Efficiency of LSMC

The effectiveness of the LSMC algorithm is evaluated from two aspects: convergence and computational efficiency. They mainly depend on the number of odd functions K, the number of discrete time points N, the number of sample paths M, and the method used to generate the simulated random numbers. Regarding the convergence of the LSMC algorithm, Longstaff and Schwartz proved that, if the number of sample paths M tends to infinity, as long as the number of basis functions is large enough, the simulated option value will converge to the actual option value. In addition, a very important feature of the LSMC algorithm is its excellent computational efficiency. Compared with the generic Monte Carlo, the computational efficiency of the algorithm is more than 600 times more efficient (Tian and Benkrid 2009).

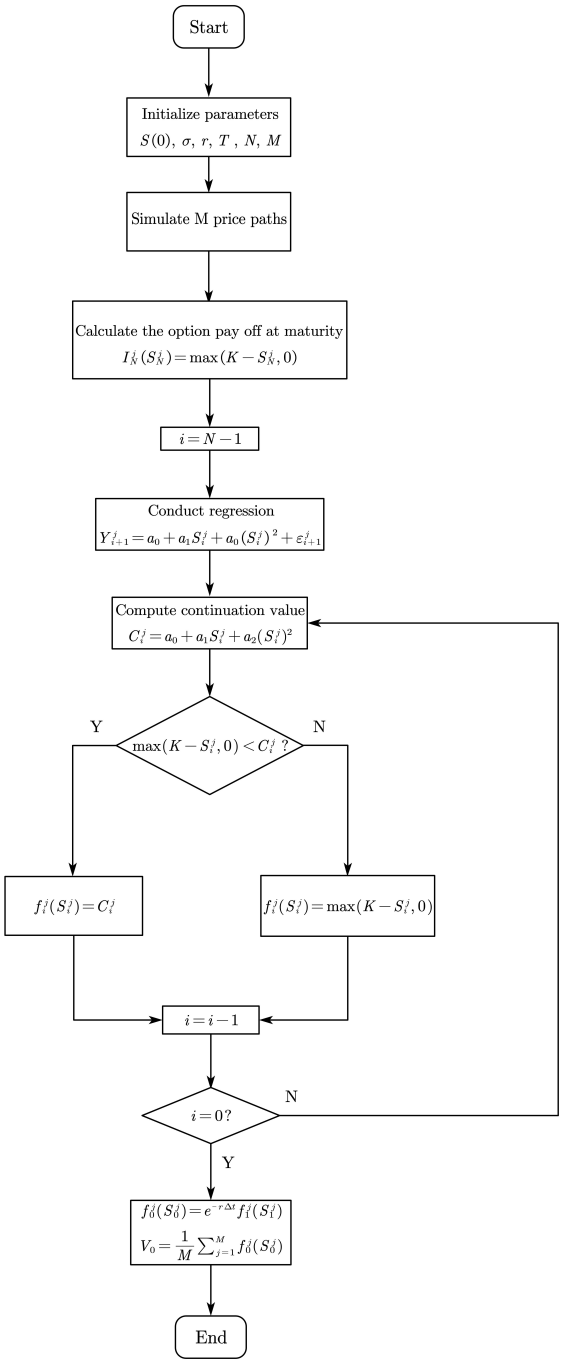

Figure 1. LSMC algorithm flow chart.

3.4. Limitations of LSMC

While LSMC simulation holds the benefit of accuracy, it is subject to some criticism and has limitations. Its performance is possibly affected by the choice in regressors. The number of regressors in the LSMC model increases exponentially when there are multiple risk factors and higher degree terms included in the basis function. This can dramatically slow the computation. Also, the LSMC method is often unsatisfactory in situations involving non-linearity. In fact, the original LSMC method's performance depends on the choice of the appropriate basis variables and their associated functions, which requires the correct selection of basis functions to ensure the accuracy of estimation as well as the most accurate number of simulation paths. However, a polynomial as the base function provides an accurate estimation of the option price regardless.

Financial institutes face exposure to multiple risk factors, such as market prices of financial securities, long-term interest rates, exchange rates, and prices inflation, etc. (Lin et al. 2018). When the LSMC model is applied to financial practice, usually, dozens of risk factors are used as regressors to estimate the conditional expectation in this model. When multiple risk factors are included in the LSMC model, the number of regressors in the regression step increases exponentially, which would make the computation slow. On the other hand, the degree could be very high, and the relationship between the independent variable and the dependent variable may be complex and non-linear when using machine learning. When there is complex and non-linear interaction between the independent and dependent variables, using the higher-degree regressors can help to capture the complexity in these relationships. The number of regressors can be determined by the following formula, where D is the degree of the regression equation and R is the number of risk factors:

$$\text{numberofregressors} = \sum_{i=1}^{D} \frac{(i+R-1)!}{i!(R-1)!} \tag{12}$$

Table 1 provides an illustrative example of the relationship between the number of regressors with different degrees and the number of risk factors based on Formula (12).

Table 1. Number of regressors with different degrees and number of risk factors.

$D \times R$	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8
2	2	5	9	14	20	27	35	44
3	3	9	19	34	55	83	119	164
4	4	14	34	69	125	209	329	494
5	5	20	55	125	251	461	791	1286
6	6	27	83	209	461	923	1715	3002
7	7	35	119	329	791	1715	3431	6434
8	8	44	164	494	1286	3002	6434	12,869

4. Distributed LSMC

4.1. Distributed Regression

To further speed up LSMC, we introduce distributed regression and MapReduce to LSMC. In 2015, Zhang et al. (2015) proposed a simple yet powerful distributed regression algorithm. This algorithm divides the entire dataset into multiple subsets, applies regression to each subset data, and then averages the results. We can distribute all the paths in LSMC across multiple computing nodes and average the results computed from all the nodes to obtain the final price of the option. Assume M is the number of paths, N is the number of time steps, K is the number of basis functions used in the regression, and m is the number of distributed computing cores. This process can be implemented using MapReduce, which is a divide and conquer method, and this process is efficient with computational complexity between $O(MNK^2/\mathrm{m})$ and $O(MNK^2)$ compared with ordinary regression LSMC with computational complexity $O(MNK^2)$.

The idea of distributed LSMC is illustrated in Figure 2. The entire dataset P is partitioned into m subsets (usually of equal size), and these subsets are labeled as $P_1, P_2, \cdots, P_m$ and then distributed to each machine. The parameter server collects the estimated option price $g(P_i)$ obtained by machine i from the subset, $i = 1, 2, \cdots, m$, and then the parameter server computes their average value $g(P)$, which yields the final estimated option price.

The primary advantage of employing distributed computing in LSMC is its ability to significantly reduce the total computational complexity of the algorithm by distributing the workload across multiple nodes, which is called the divide and conquer method. This can lead to faster computation times and improved scalability, allowing for the efficient pricing of American options and even large portfolios of derivatives.

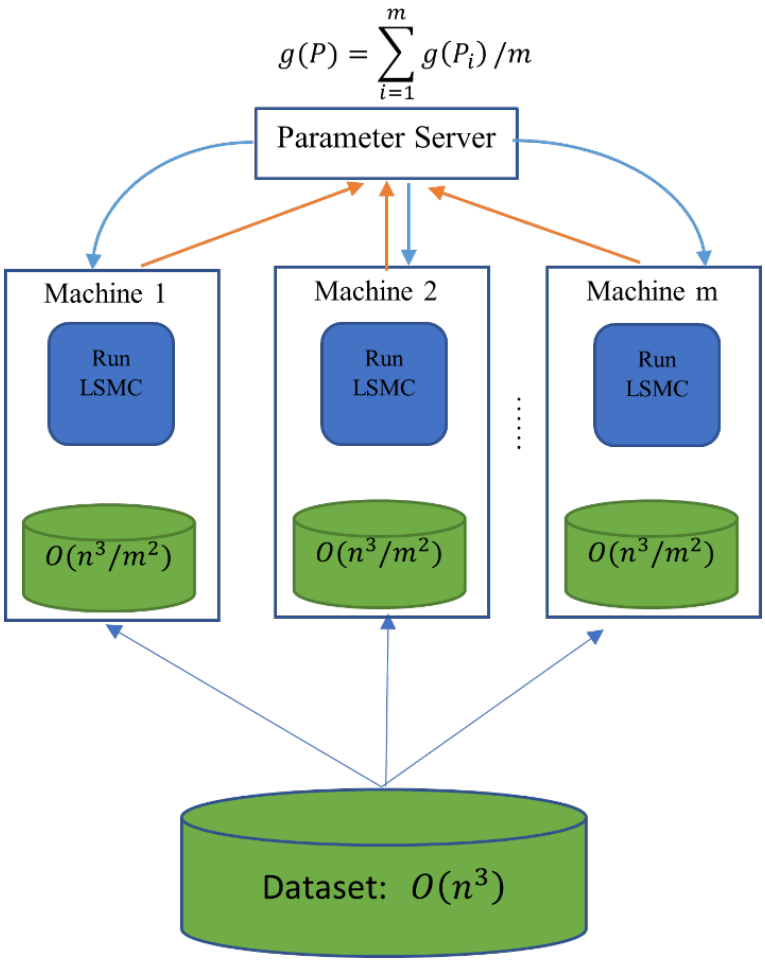

Figure 2. The idea of distributed LSMC.

Additionally, distributed computing can provide fault tolerance and increased reliability by allowing for redundant computations and reducing the risk of system failures or crashes.

Finally, distributed computing can protect data privacy by using data fragmentation, secure communication protocol, and other technologies such as encryption. By breaking the data into smaller subsets and distributing them into multiple computers, each computer has its own security measures. By doing this, it is more difficult for hackers to access whole datasets because they must compromise multiple systems simultaneously.

4.2. Validation of Distributed LSMC

Inspired by distributed regression, we propose the distributed LSMC to reduce the runtime of LSMC.

Figure 3 shows the flow chart of the distributed LSMC as described in Algorithm 1.

In the following paragraphs, we will discuss the validity of the proposed distributed LSMC algorithm. We will focus on explaining why this algorithm can obtain the same result as the traditional LSMC. Firstly, the LSMC method, like other Monte Carlo methods, is based on the Law of Large Numbers (Graham et al. 2013). As the sample size increases, the sample average converges to the population expectation. Therefore, if we simulate the price paths and calculate the payoff of the option using LSMC independently on each computer node, we can expect the estimated value from each node will converge to the same result with enough simulated paths. Secondly, in our proposed distributed algorithm, each path on the distributed computer node is mutually independent and the asset returns follow the same probability distribution. Therefore, when we average these independently

estimated option payoffs, we can obtain a stable and accurate estimation of the actual option price.

Algorithm 1: Algorithm of Distribution LSMC

1 Step 1: Set the total number of simulated paths N.
2 Step 2: Determine the number of distributed computing cores m.
3 Step 3: Equally divide the N simulated paths into m subsets, each containing N/m paths. Label these subsets as $P_1, P_2, \cdots, P_m$.
4 Step 4: On each computing core i, independently generate path subset P_i and apply the LSMC algorithm to P_i to obtain the estimated option price $g(P_i)$. In this step, the LSMC algorithm used on each core is the same as the conventional LSMC method.
5 Step 5: Compute the average of all the estimated option prices from the computing cores as the final estimated option price: $[g(P_1) + g(P_2) + \ + g(P_m)]/m$.

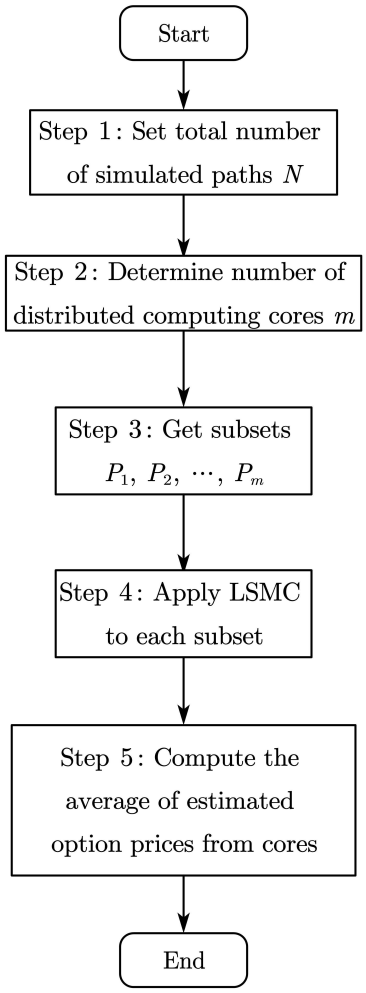

Figure 3. Distributed LSMC algorithm flow chart.

Theorem 1. *The expected result from the distributed LSMC method is identical to the expected result from the traditional LSMC method, i.e.,*

$$E[g(P_1 \cup P_2 \cup \cdots \cup P_m)] = E\left[\sum_{i=1}^{m} \frac{g(P_i)}{m}\right]. \tag{13}$$

Proof. LHS: $E[g(P_1 \cup P_2 \cup \cdots \cup P_m)]$ is the expectation of the option payoff calculated by LSMC over all paths; according to Longstaff and Schwartz (2001), this equals the true option price V.

RHS: Note that each P_i is an independent subset of all paths; hence, the expected option price by LSMC for each subset is also V. Therefore, $E[\sum_{i=1}^{m} g(P_i)/m] = \sum_{i=1}^{m} \frac{E[g(P_i)]}{m} = \frac{mV}{m} = V$.

Hence, $E[g(P_1 \cup P_2 \cup \cdots \cup P_m)] = E[\sum_{i=1}^{m} g(P_i)/m]$. □

This theorem proved that both methods produce the same expected results. Also, because both the traditional and distributed LSMC will converge to their expected estimation of the option price, they will converge to the same results. Therefore, Theorem 1 proved the equivalence of the results produced by both methods.

Several researchers have applied parallel regression in LSMC. Parallel computing for regression problems often involves more complicated matrix computation. Especially in the least square linear regression, an effective parallel algorithm is using matrix decomposition techniques such as QR decomposition or Cholesky decomposition (Datta 1985). The distributed LSMC has advantages over the parallel LSMC in terms of computational complexity and scalability. By avoiding the complex matrix computation in the parallel LSMC, the distributed LSMC reduces the total computational complexity by partitioning the data into smaller subsets. In addition, this strategy naturally fits the architecture of cloud computing, which is built on distributed computing technologies. With enough cloud computing resources, we can greatly reduce the computing time required and provide advantages in application environments where efficiency and scalability are critical (Leopold 2001). It is essential to distinguish between these two approaches (Riesen et al. 1998). A parallel computing system consists of multiple processors that communicate with each other using a shared memory, as shown in Figure 4, whereas a distributed computing system contains multiple processors connected by a communication network, as shown in Figure 5. The parallel regression requires complex matrix transformations, while the distributed LSMC method we proposed here does not require that (Datta 1985).

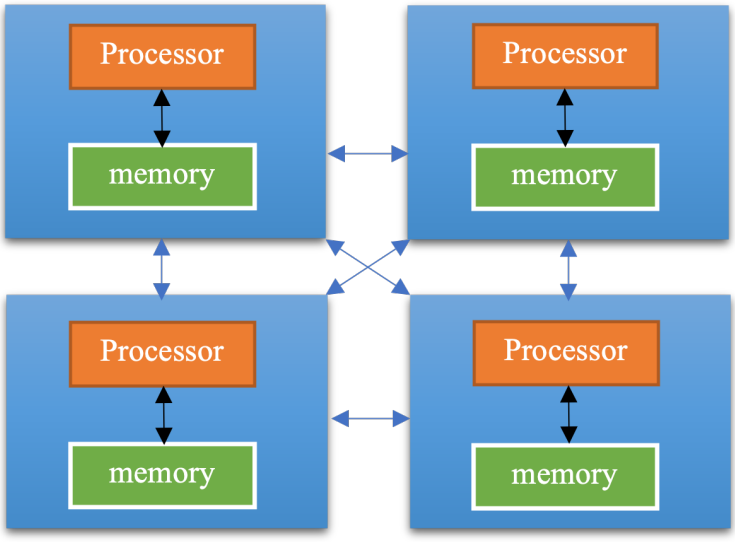

Figure 4. The idea of distributed computing.

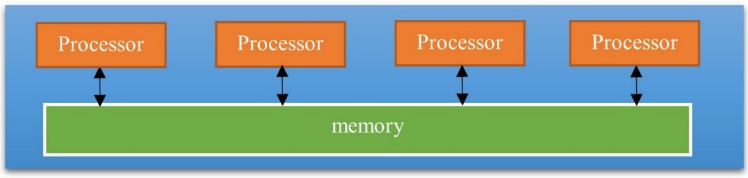

Figure 5. The idea of parallel computing.

5. Computational Complexity Comparison

Computational complexity is a branch of theoretical computer science that focuses on the time and space complexity of algorithms. Big O notation is often used to compare different algorithms and to choose the most efficient algorithm for a given problem (Devi et al. 2011).

The following is the complexity analysis of some American options pricing approaches.

Assume M is the number of paths, N is the number of time steps, K is the number of basis functions used in the least squares regression, m is the number of distributed computing cores, and M^* represents the number of scenarios corresponding to each scenario at the previous time step.

For the binomial tree approach, the computational complexity is $O(N^2)$. This is because the tree has time steps N, and, at time $t(0 \leq t \leq N)$, there are $t+1$ nodes. Therefore, the total number of nodes in the binomial tree is $1+2+\cdots+N+(N+1) = (N+1)(N+2)/2$, so the computational complexity is $O(N^2)$ if dismissing other terms with smaller power.

For the generic Monte Carlo (also known as nested Monte Carlo), assume M^* represents the number of considered possible scenarios corresponding to each scenario at the previous time step (except from those of the last level), and N represents time steps. We only focus on the number of scenarios in the last level, which is M^*. Thus, the computational complexity is $O(M^*)$.

The ternary tree is a special case of the generic Monte Carlo method, where $M^* = 3$; therefore, the computational complexity is $O(3^N)$.

In the LSMC method, the regression at each time level involves the building of a square and symmetric matrix via the system of normal equations. The least squares solution can be computed using efficient methods such as Cholesky factorization, which has a computational complexity of $O(K^3)$. However, the construction of the matrix A^T A has complexity $O(MK^2)$ (Stothers 2010) based on the complexity of matrix multiplication. In typical data fitting problems, $M \gg N \gg K$, and, hence, the overall complexity of the normal equations method (Herzberger 1949) is $O(MK^2)$. Considering that there are N time steps, the computational complexity of LSMC is $O(MNK^2)$.

The distributed LSMC involves the simulation of the price paths, least squares regression for the continuation value, and comparison between the early exercise value and the continuation value, etc. In the regression step, the distributed LSMC can reduce the overall computational complexity to $m * O((MNK^2)/\mathrm{m}^2)$ but does not change the complexity of other steps. Therefore, the computational complexity of the distributed LSMC will be lower than the traditional LSMC $O(MNK^2)$ but higher than the pure distributed regression $O(MNK^2/\mathrm{m})$.

Since

$$O\left(MNK^2/m\right) < O\left(MNK^2\right) < O\left(M^{*N}\right) < O\left(3^N\right) < O\left(N^2\right) \tag{14}$$

We can rank the computational complexities of these methods as *Distributed LSMC* < *LSMC* < *Binomial Tree* < *Ternary Tree* < *Generic Monte Carlo*. We put these results in Table 2.

Table 2. Computational complexity of different approaches.

Pricing Approaches	Computational Complexity
Binomial Tree	$O\left(N^2\right)$
Ternary Tree	$O\left(3^N\right)$
Generic Monte Carlo	$O\left(M^{*N}\right)(N \geq 3)$
LSMC	$O\left(MNK^2\right)$
Distributed LSMC	Between $O\left(MNK^2/\mathrm{m}\right)$ and $O\left(MNK^2\right)$

6. Experiment: Result of Distributed Regression Method for LSMC

In this chapter, we aim to conduct a series of computing experiments for the distributed LSMC we have proposed. We will evaluate the computing speed and accuracy of the distributed LSMC vs the traditional LSMC. The main goal of this comparison research is to validate the theory we derived, that these two methods would produce equivalent and very close results.

When it comes to computing speed, we anticipate that the distributed LSMC method provides a significant performance improvement. This anticipation of the efficiency increase comes from the intrinsic advantage of distributed computing, which uses the computing resources of multiple machines to reduce the total computing time. On the other hand, the distributed LSMC can reduce the total computational complexity, which results in shorter computing time required. This chapter will discuss the experimental study results to support these statements.

Our experiment was carried out on Google CoLab equipped with the following hardware specifications:

- CPU Model Name: Intel(R) Xeon(R)
- CPU Frequency: 2.30 GHz
- Number of CPU Cores: 2
- CPU Family: Haswell
- Available RAM: 12 GB
- Disk Space: 25 GB

There are two popular frameworks to implement distributed computing: Hadoop and Spark. The bottleneck of the Hadoop MapReduce is unnecessary data reading and writing, and this is the main improvement in Spark. Specifically, Spark continues the design idea of Hadoop: the calculation of data is also divided into two types: Map and Reduce. However, the difference is that a Spark task does not only include a Map and a Reduce but consists of a series of Maps and Reduces. In this way, the intermediate results of the calculation can be efficiently transferred to the next calculation step to improve the performance of the algorithm (Fu et al. 2016). Although the improvement in Spark seems to be small, the experimental results show that its algorithm performance is 10–100 times higher than that of Hadoop (Mostafaeipour et al. 2021).

Because of the advantages of Spark, we use PySpark 3.3.1 and Python 3.9.7 for distributed LSMC implementation.

The following is detailed information about the American option example we are going to price.

- Initial underlying asset price: S_0 = USD 100.
- Strike Price: K = USD 101, K = USD 200.
- Time to expiration: $T = 10$ "days". This is the length of time during which the American option contract is valid.
- Annualized interest rate: $r = 0.05$.
- Annualized volatility of underlying asset return: $\sigma = 30\%$.
- Dividends: $\delta = 0$. Assume no dividend is paid during the life of the option contract.

The benchmark values for the above American options, with strike prices of K = USD 101 and USD 200, are USD 2.455145310460293 and USD 100, respectively. They will be used as the benchmark to compare our computing results.

Please refer to Table 3 and Table 4 for our speed experiment results. We used two computer cores on Google CoLab and tried different numbers of paths ranging from 1 million to 10 million. All the computations were performed in double precision to ensure a high degree of accuracy when comparing the small errors. From these results, we found that the distributed LSMC is consistently faster than the traditional LSMC. In addition, with the number of paths increasing, the percentage of time the LSMC method saves also increases. For instance, in Table 3, when considering 1 million paths, the distributed LSMC is about 65% faster than the traditional LSMC. With 10 million paths, the improvement becomes

more significant, where the distributed LSMC is nearly three times faster. This phenomenon could be due to the startup time by the PySpark when initiating distributed computing. As the number of computing tasks and the amount of data being processed increase, this fixed amount of starting time becomes a smaller fraction of the total computing time. Therefore, as the simulated paths increase, the relative speed advantage of the distributed LSMC becomes more obvious. These results confirmed the advantages of the distributed LSMC method in large-scale computing tasks.

The accuracy of our distributed LSMC compared with traditional LSMC is listed in Table 5 and Table 6. The second and third columns of these tables show the pricing results of two algorithms with different numbers of simulated paths. The last two columns are percentage errors of the pricing results by comparing two algorithms with the actual option price. This result shows that the distributed LSMC can achieve comparable pricing accuracy to the traditional LSMC. As the number of paths increases, both methods produce more accurate results. Thus, we conclude that our distributed LSMC can achieve faster computing speed than the traditional LSMC without sacrificing accuracy.

Table 3. Algorithm speed experiment on Google Colab with Apache Spark (strike price K = USD 101, 2 cores, set number of machines = 2, s stands for seconds).

Paths	Distributed LSMC	LSMC
100	0.2710 s	0.0438 s
1000	0.3713 s	0.0518 s
10,000	0.2813 s	0.0723 s
100,000	0.6980 s	0.2958 s
1,000,000	7.2566 s	11.9770 s
2,000,000	10.2317 s	24.7639 s
3,000,000	14.6611 s	35.1827 s
4,000,000	19.4818 s	48.9408 s
5,000,000	26.0488 s	60.8501 s
6,000,000	30.5395 s	77.8814 s
7,000,000	36.2866 s	97.0932 s
8,000,000	39.9882 s	121.1392 s
9,000,000	44.4849 s	138.0011 s
10,000,000	53.1129 s	152.7216 s

Table 4. Algorithm speed experiment on Google Colab with Apache Spark (strike price K = USD 200, 2 cores, set number of machines = 2, s stands for seconds).

Paths	Distributed LSMC	LSMC
100	0.2748 s	0.0459 s
1000	0.2755 s	0.0575 s
10,000	0.3889 s	0.1073 s
100,000	0.4837 s	0.3910 s
1,000,000	6.2949 s	4.2849 s
2,000,000	6.3678 s	13.8650 s
3,000,000	10.5806 s	13.082 s
4,000,000	14.5165 s	21.3278 s
5,000,000	18.9077 s	26.9697 s
6,000,000	22.6712 s	30.3915 s
7,000,000	26.438 s	35.5304 s
8,000,000	30.161 s	40.4518 s
9,000,000	33.4855 s	45.9881 s
1,000,0000	38.1134 s	51.1145 s

Table 5. Algorithm accuracy experiment on Google Colab with Apache Spark (strike price K = USD 101, 2 cores, set number of machines = 2).

Paths	Distributed LSMC	Price Using LSMC	Error (Distributed LSMC)	Error (LSMC)
100	$2.7277	$2.5374	11.101%	3.350%
1000	$2.4391	$2.3996	0.654%	2.262%
10,000	$2.4283	$2.4315	1.093%	0.963%
100,000	$2.4427	$2.4541	0.507%	0.043%
1,000,000	$2.4586	$2.4573	0.141%	0.088%
2,000,000	$2.4549	$2.4532	0.010%	0.079%
3,000,000	$2.4502	$2.4568	0.201%	0.067%
4,000,000	$2.4515	$2.4551	0.148%	0.002%
5,000,000	$2.4534	$2.4561	0.071%	0.039%
6,000,000	$2.4541	$2.4560	0.043%	0.035%
7,000,000	$2.4534	$2.4564	0.071%	0.051%
8,000,000	$2.4531	$2.4552	0.083%	0.002%
9,000,000	$2.4541	$2.4560	0.043%	0.035%
1,000,0000	$2.4537	$2.4562	0.059%	0.043%

Table 6. Algorithm accuracy experiment on Google Colab with Apache Spark (strike price K = USD 200, 2 cores, set number of machines = 2).

Paths	Distributed LSMC	Price Using LSMC	Error (Distributed LSMC)	Error (LSMC)
100	$100.4990	$100.3112	0.499%	0.311%
1000	$99.9805	$99.9487	0.019%	0.051%
10,000	$99.9429	$99.9578	0.057%	0.042%
100,000	$99.9667	$99.9656	0.033%	0.034%
1,000,000	$99.9676	$99.9716	0.032%	0.028%
2,000,000	$99.9709	$99.9729	0.029%	0.027%
3,000,000	$99.9702	$99.9728	0.030%	0.027%
4,000,000	$99.9716	$99.9723	0.028%	0.028%
5,000,000	$99.9722	$99.9728	0.028%	0.027%
6,000,000	$99.9720	$99.9726	0.028%	0.027%
7,000,000	$99.9725	$99.9729	0.028%	0.027%
8,000,000	$99.9729	$99.9728	0.027%	0.027%
9,000,000	$99.9729	$99.9728	0.027%	0.027%
1,000,0000	$99.9726	$99.9727	0.027%	0.027%

7. Conclusions

This paper has explored the application of distributed computing technology in enhancing the LSMC method for American option pricing. Option pricing is a critical area of research in the financial market, and the efficiency and accuracy of pricing solutions play a significant role in the stability and development of the market. While computational techniques, including parallel computing, have been widely utilized to optimize pricing algorithms, this paper stands out as the first to investigate the potential of distributed computing technology for LSMC enhancement in American option pricing.

Compared to parallel computing, distributed computing offers several advantages. By employing a "divide and conquer" approach, it reduces computational complexity, enabling faster and more efficient calculations. Moreover, the distributed computing approach eliminates the need for complicated matrix transformations and enhances data privacy and security. These advantages make distributed computing a suitable and promising technique for LSMC in American option pricing. The LSMC method itself aligns well with the distributed computing paradigm since price paths can be simulated and regressed separately.

To validate the correctness of the distributed LSMC algorithm we proposed, we proved the equivalency of distributed and traditional LSMC methods in terms of pricing expectations in probability theory. In addition, we analyzed the computational complexity of some American option pricing methods: binomial tree, ternary tree, generic Monte Carlo,

traditional LSMC, and the distributed LSMC proposed in this paper. The distributed LSMC has the least computational complexity.

The primary objective of this research was to showcase how distributed computing can significantly improve the accuracy and efficiency of LSMC in American option pricing. For this reason, we conducted numerical experiments for both the traditional and distributed LSMC, which verified the accuracy and efficiency of the distributed LSMC.

Other than the application in American option pricing, the distributed LSMC method proposed in this paper also has broad application in the insurance industry. This method works for the insurance products that contain the American option or those similar to the American option. For instance, the variable annuity includes the embedded American style options for the policyholder, and its fair market value depends on the performance of the underlying investment portfolio. The distributed LSMC can provide efficient and accurate pricing results for these types of products that are complex and path-dependent.

However, there are certain limitations of the study worth mentioning. Firstly, we only explored polynomial functions as base functions with degree 2, whereas there are many other types of base functions available, such as triangle functions, spline functions, and wavelet functions. Further, the degree could be increased. Optimizing the choosing of base function and degree could improve the performance of the algorithm. Secondly, we conducted the experiment using only two CPU cores, without increasing the number of cores. Thirdly, distributed computing involves communication between different computing nodes, which can introduce overhead due to data transfer and synchronization. The increased communication overhead can reduce the potential speedup gained from parallel processing. Lastly, ensuring an even distribution of computational tasks among the nodes can be challenging. Variability in option pricing complexities and data distribution might lead to load imbalance, where some nodes are underutilized while others are overwhelmed, which can hinder overall performance gains.

In the future, we plan to conduct some additional works. Firstly, we will apply distributed LSMC in the field of insurance for American option-type products, specifically with variable annuities. Secondly, we aim to explore the application of machine learning methods to estimate the discounted continuation values instead of relying on a regression model. Thirdly, we will endeavor to further accelerate PySpark calculations using GPUs.

In conclusion, our exploration of distributed computing technology in the context of LSMC for American option pricing presents a valuable contribution to the financial market. The findings and insights presented in this paper pave the way for improved pricing solutions and hold the potential to drive significant progress in the field of American option pricing research.

Author Contributions: Conceptualization, L.X.; methodology, L.X. and J.L.; software, L.X. and J.L.; validation, L.X. and J.L.; writing—original draft preparation, L.X., J.L., H.V. and M.W.; writing—review and editing, L.X., J.L., H.V. and M.W.; visualization, L.X. and J.L.; supervision, L.X.; project administration, L.X.; funding acquisition, L.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data can be made available on request to the corresponding author.

Acknowledgments: We would like to express our gratitude for the editors and the reviewers of Risks for their work.

Conflicts of Interest: The authors declare no conflict of interest.

References

Bauer, Daniel, Daniela Bergmann, and Andreas Reuss. 2010. Solvency ii and nested simulations—A least-squares monte carlo approach. In *Proceedings of the 2010 ICA Congress*. Sydney: Citeseer.

Black, Fischer, and Myron Scholes. 1973. The pricing of options and corporate liabilities. *Journal of Political Economy* 81: 637–54. [CrossRef]

Chen, Ching-Wen, Kuan-Lin Huang, and Yuh-Dauh Lyuu. 2015. Accelerating the least-square monte carlo method with parallel computing. *The Journal of Supercomputing* 71: 3593–608. [CrossRef]
Chen, Yangang, and Justin W. L. Wan. 2021. Deep neural network framework based on backward stochastic differential equations for pricing and hedging american options in high dimensions. *Quantitative Finance* 21: 45–67. [CrossRef]
Clément, Emmanuelle, Damien Lamberton, and Philip Protter. 2002. An analysis of a least squares regression method for american option pricing. *Finance and Stochastics* 6: 449–71. [CrossRef]
Cox, John C., Stephen A. Ross, and Mark Rubinstein. 1979. Option pricing: A simplified approach. *Journal of Financial Economics* 7: 229–63. [CrossRef]
Dai, Tian-Shyr, and Yuh-Dauh Lyuu. 2010. The bino-trinomial tree: A simple model for efficient and accurate option pricing. *The Journal of Derivatives* 17: 7–24. [CrossRef]
Datta, Karabi. 1985. Parallel complexities and computations of cholesky's decomposition and qr factorization. *International Journal of Computer Mathematics* 18: 67–82. [CrossRef]
Devi, S. Gayathri, K. Selvam, and S. P. Rajagopalan. 2011. An abstract to calculate big o factors of time and space complexity of machine code. Paper presented at the International Conference on Sustainable Energy and Intelligent Systems (SEISCON 2011), Chennai, India, July 20–22.
Dobriban, Edgar, and Yue Sheng. 2021. Distributed linear regression by averaging. *arXiv* arxiv:1810.00412. [CrossRef]
Fu, Jian, Junwei Sun, and Kaiyuan Wang. 2016. Spark–a big data processing platform for machine learning. Paper presented at the 2016 International Conference on Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII), Wuhan, China, December 3–4, pp. 48–51.
Goudenège, Ludovic, Andrea Molent, and Antonino Zanette. 2019. Variance reduction applied to machine learning for pricing bermudan/american options in high dimension. *arXiv* arXiv:1903.11275.
Graham, Carl, Denis Talay, Carl Graham, and Denis Talay. 2013. Strong law of large numbers and monte carlo methods. In *Stochastic Simulation and Monte Carlo Methods: Mathematical Foundations of Stochastic Simulation*. Berlin/Heidelberg: Springer, pp. 13–35.
Haugh, Martin B., and Leonid Kogan. 2004. Pricing american options: A duality approach. *Operations Research* 52: 258–70. [CrossRef]
Herzberger, M. 1949. The normal equations of the method of least squares and their solution. *Quarterly of Applied Mathematics* 7: 217–23. [CrossRef]
Krah, Anne-Sophie, Zoran Nikolić, and Ralf Korn. 2018. A least-squares monte carlo framework in proxy modeling of life insurance companies. *Risks* 6: 62. [CrossRef]
Leopold, Claudia. 2001. *Parallel and Distributed Computing: A Survey of Models, Paradigms and Approaches*. Hoboken: John Wiley & Sons, Inc.
Lin, Edward M. H., Edward W. Sun, and Min-Teh Yu. 2018. Systemic risk, financial markets, and performance of financial institutions. *Annals of Operations Research* 262: 579–603. [CrossRef]
Longstaff, Francis A., and Eduardo S. Schwartz. 2001. Valuing american options by simulation: a simple least-squares approach. *The Review of Financial Studies* 14: 113–47. [CrossRef]
Mostafaeipour, Ali, Amir Jahangard Rafsanjani, Mohammad Ahmadi, and Joshuva Arockia Dhanraj. 2021. Investigating the performance of hadoop and spark platforms on machine learning algorithms. *The Journal of Supercomputing* 77: 1273–300. [CrossRef]
Riesen, Rolf, Ron Brightwell, and Arthur B. Maccabe. 1998. *Differences between Distributed and Parallel Systems*. SAND98-2221, Unlimited Release, Printed October. Available online: https://www.osti.gov/biblio/1518 (accessed on 12 July 2023).
Stothers, Andrew James. 2010. On the complexity of matrix multiplication. In *Edinburgh Research Archive*. Edinburgh: The University of Edinburgh.
Tian, Xiang, and Khaled Benkrid. 2009. American option pricing on reconfigurable hardware using least-squares monte carlo method. Paper presented at the 2009 International Conference on Field-Programmable Technology, Sydney, Australia, December 9–11, pp. 263–70.
Tilley, James A. 1993. Valuing american options in a path simulation model. *Transactions of the Society of Actuaries* 45: 499–519.
Weigel, Martin. 2018. Monte carlo methods for massively parallel computers. In *Order, Disorder and Criticality: Advanced Problems of Phase Transition Theory*. Singapore: World Scientific, pp. 271–340.
Zhang, Yuchen, John Duchi, and Martin Wainwright. 2015. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *The Journal of Machine Learning Research* 16: 3299–40.