**Monte Carlo Simulation for Predicting Bitcoin Price Using Geometric Brownian Motion Powered By High Performance Computing**

**Problem statement:**

The rapid fluctuations in Bitcoin prices pose significant challenges in forecasting and risk assessment. Traditional deterministic models fail to capture the inherent randomness in cryptocurrency markets, making stochastic approaches more suitable. Geometric Brownian Motion (GBM) provides a widely accepted mathematical framework for modelling asset price dynamics, incorporating both drift and volatility. By integrating GBM with Monte Carlo Simulation (MCS), it is possible to generate a distribution of potential future Bitcoin prices rather than a single point estimate, enabling better insights into uncertainty and risk. However, Monte Carlo methods require running millions of independent simulations to achieve statistical reliability, which is computationally expensive when implemented sequentially. This project addresses the problem by first developing a sequential C++ implementation of GBM-based Monte Carlo simulation for Bitcoin price prediction, and then extending it to a parallel implementation using High Performance Computing (HPC) techniques such as OpenMP and MPI. The goal is to evaluate the scalability and efficiency of HPC methods while demonstrating their necessity in handling large-scale, compute-intensive financial simulations.



**Methodology:**

The methodology adopted in this project is grounded in stochastic modeling of financial time series, specifically through the use of Geometric Brownian Motion (GBM), and computationally intensive simulation via the Monte Carlo Simulation (MCS) framework. The process begins with historical Bitcoin data collection and preprocessing, which provides the foundation for estimating the parameters of GBM. Bitcoin’s log-returns are computed from the dataset, and statistical measures such as mean and standard deviation are used to estimate the drift μ and volatility σ of the model. These parameters are critical because drift captures the average growth trend of Bitcoin, while volatility quantifies its randomness and instability.

Once the GBM parameters are determined, they are incorporated into the stochastic differential equation of GBM:

*dS t ​ =μS t ​ dt+σS t ​ dW t ​*

which describes how the price evolves over time. In practice, this continuous equation is discretized using numerical methods, often the Euler–Maruyama scheme, to allow iterative simulation of price paths in a computer program. This discretization ensures that each simulation step captures both the deterministic component (drift) and the random shock component (volatility). By doing so, GBM produces realistic synthetic trajectories of Bitcoin prices that preserve positivity and reflect the erratic nature of the cryptocurrency market.

With the GBM framework in place, Monte Carlo Simulation (MCS) is employed to generate a large ensemble of possible Bitcoin price paths. Each path represents a potential evolution of the asset’s value under the influence of random shocks. By running thousands or even millions of such paths, the method produces a statistical distribution of possible future prices at a given horizon. This allows not only the computation of the expected value of Bitcoin’s price but also the extraction of more sophisticated insights such as confidence intervals, volatility forecasts, and risk measures like Value-at-Risk (VaR). The strength of MCS lies in its ability to quantify uncertainty, making it especially relevant for volatile assets like Bitcoin where single deterministic forecasts are unreliable.

The project is executed in two stages: non-HPC (sequential) and HPC (parallel) implementations. In the sequential version, a C++ program is developed to read historical data, estimate GBM parameters, and run the Monte Carlo simulation loop in a single-threaded environment. This provides a correct and transparent baseline, enabling verification of the methodology and results. However, the computational cost of this approach grows linearly with the number of simulations, and for large NNN, the runtime quickly becomes impractical. For instance, while simulating 10^5 paths may be feasible in seconds, scaling up to 10^8 paths may take hours or days on a single processor core.

A white rectangular sign with black text

AI-generated content may be incorrect.

Figure 1:Montecarlo simulation for bitcoin price prediction using Geometric Brownian Motion without HPC

A screenshot of a diagram

AI-generated content may be incorrect.

Figure 2:Montecarlo simulation for bitcoin price prediction using Geometric Brownian Motion HPC

To overcome these scalability limitations, the methodology transitions to a **High Performance Computing (HPC)** paradigm. The first extension is carried out using **MPI)**. In this implementation, the simulation workload is divided across multiple processes running on different nodes of a cluster or across multiple CPUs. Each MPI process independently generates a subset of Monte Carlo simulation paths, ensuring statistical independence while distributing the computational load. The results from all processes are then combined using collective communication primitives. This distributed-memory model enables the system to scale to very large numbers of simulations by harnessing multiple computing nodes, thereby drastically reducing runtime compared to the sequential version.

Once the MPI-based distributed version is established, the project further enhances performance by incorporating **OpenMP** for intra-node parallelism. Within each MPI process, OpenMP is used to parallelize the simulation loop across multiple threads, taking advantage of the shared-memory architecture of modern multicore processors. This hybrid MPI + OpenMP approach allows simulations to exploit both inter-node parallelism (via MPI) and intra-node parallelism (via OpenMP). As a result, the computational capacity is maximized, enabling the efficient execution of billions of simulation paths, which is necessary for statistically reliable Bitcoin price prediction and risk analysis.