Parallelizing Computation of Expected Values in Recombinant Binomial Trees

Sai K. Popuri[a], Andrew M. Raim[b], Nagaraj K. Neerchal[a], and Matthias K. Gobbert[a]

[a]Department of Mathematics and Statistics, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, USA; [b]Center for Statistical Research & Methodology, U.S. Census Bureau, 4600 Silver Hill Road, Washington, DC 20233, USA

ABSTRACT
Recombinant binomial trees are binary trees where each non-leaf node has two child nodes, but adjacent parents share a common child node. Such trees arise in finance when pricing an option. For example, valuation of a financial option can be carried out by evaluating the expected value of asset payoffs with respect to random paths in the tree. In many variants of the option valuation problem, a closed form solution cannot be obtained and computational methods are needed. The cost to exactly compute expected values over random paths grows exponentially in the depth of the tree, rendering a serial computation of one branch at a time impractical. We propose a parallelization method that transforms the calculation of the expected value into an embarrassingly parallel problem by mapping the branches of the binomial tree to the processes in a multiprocessor computing environment. We also discuss a parallel Monte Carlo method which takes advantage of the mapping to achieve a reduced variance over the basic Monte Carlo estimator. Performance results from `R` and `Julia` implementations of the parallelization method on a distributed computing cluster indicate that both the implementations are scalable, but `Julia` is significantly faster than a similarly written `R` code. A simulation study is carried out to verify the convergence and the variance reduction behavior in the parallel Monte Carlo method.

KEYWORDS
Binomial tree, Bernoulli paths, Monte Carlo estimation, Option pricing.

1. Introduction

An N-step recombinant binomial tree is a binary tree where each non-leaf node has two children, which we will label "up" and "down". The tree has depth N, so that any path from the root node to a leaf node consists of N up or down steps. The tree is called recombinant because the sequence of moves (up, down) is assumed to be equivalent to the sequence (down, up). In such a tree, there are $N + 1$ distinct leaf nodes and $1 + 2 + \cdots + (N + 1) = (N + 1)(N + 2)/2$ nodes overall. Any particular path from the root to a leaf can be written as a binary sequence $\boldsymbol{x} = (x_1, \ldots, x_N)$ where $x_j \in \mathbb{B}$, $\mathbb{B} = \{0, 1\}$, and 1 corresponds to an up movement while 0 corresponds to down. Given a density $p(\boldsymbol{x}) = \mathrm{P}(\boldsymbol{X} = \boldsymbol{x})$, we may consider $\boldsymbol{X}$ as a random path from the root to a leaf. We will refer to random variables $\boldsymbol{X} \in \mathbb{B}^N$ as Bernoulli paths. An N-step binomial tree has 2^N Bernoulli paths.

A primary example of recombinant binomial trees is the binomial options pricing model proposed by Cox et al. [1]. This model accounts for uncertainty of a future stock price based on
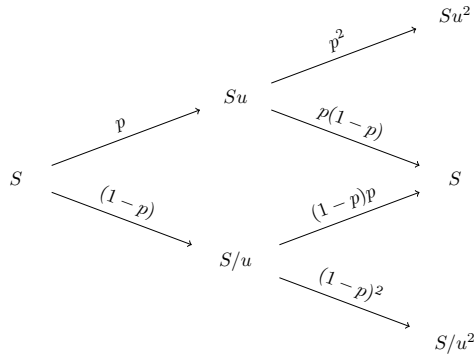
Figure 1.1.: A two-step recombinant binomial tree.

its current market price at S. Figure 1.1 illustrates a binomial options model for the evolution of the stock in $N = 2$ time periods. Starting from the root node, the stock price moves up by an amount u to Su with probability p or moves down to S/u with probability $1 - p$. After one step, each of the two child nodes further branch to two leaf nodes where a factor of u is applied with probability p or d is applied with probability $1 - p$. Here, the sequences (up, down) and (down, up) both take the stock price back to its starting price.

The binomial options pricing model is used in the valuation of financial contracts like options, which derive their value from a less complicated, underlying asset such as a stock price. In order to calculate the value of an option, one builds a recombinant binomial tree to a future time point from the current market price of the stock S using a Bernoulli probability model at each time step. Depending on the type of the option, the option value is either the present value of the expected option payoff or is calculated by traversing the tree backwards and revising the option value at each step. See Hull [2] and Seydel [3] for more details on options and their valuation. When the option payoff at a leaf node depends on the path, one must consider all 2^N possible paths to calculate the expected value of the option payoff.

Pattern-mixture models for missing longitudinal data provide a second example involving recombinant binomial trees. A brief overview is given here, while the remainder of the paper focuses on the options pricing application. In a pattern-mixture model [4], longitudinal data with missing values is available for each subject and the conditional distribution of the data given the pattern of missingness is considered. Let Y_{it} be the response from subject i at time t, where $i = 1, \ldots, n$ and $t = 1, \ldots, T$. The multivariate response $\boldsymbol{Y}_i = (Y_{i1}, \ldots, Y_{iT})$ may contain missing data whose pattern is denoted by $\boldsymbol{Z}_i = (Z_{i1}, \ldots, Z_{iT})$; Z_{it} is 0 if Y_{it} is observed and 1 if missing. The joint distribution of the observed $\{(\boldsymbol{Y}_i, \boldsymbol{Z}_i) : i = 1, \ldots, n\}$ for such a model is given by

$$\prod_{i=1}^{n} f(\boldsymbol{y}_i \mid \boldsymbol{z}_i, \boldsymbol{\theta}) g(\boldsymbol{z}_i \mid \boldsymbol{\theta}), \tag{1.1}$$

where f and g are the probability functions of $\boldsymbol{y}_i \mid \boldsymbol{z}_i$ and $\boldsymbol{z}_i$, respectively. Note that the expected value calculations with respect to $\boldsymbol{z}$ will involve summing over all Bernoulli paths $\boldsymbol{z}$.

In applications of recombinant binomial trees, such as the two previously mentioned, it is often required to compute the expected value of a function $V(\boldsymbol{X})$

$$\mathrm{E}[V(\boldsymbol{X})] = \sum_{\boldsymbol{x} \in \mathbb{B}^N} V(\boldsymbol{x}) p(\boldsymbol{x}). \tag{1.2}$$

The option value calculation and the pattern-mixture likelihood (1.1) both take this form. The function $V(\boldsymbol{x})$ may depend on the entire path $\boldsymbol{x}$, and not only on the leaf nodes. Notice that (1.2)

is a summation over 2^N terms, so that computing by complete enumeration quickly becomes infeasible as N increases. In this work, we present a method to parallelize the calculation in a multiprocessor computing environment. In the option valuation problem, the common method to value options is to use an efficient backward induction method without considering the 2^N terms in (1.2). The proposed parallelization method is suitable for advanced class of path-dependent options that are valued by sampling paths off the recombinant binomial tree than through backward induction [5, Chapter 4.]. Our method uses a Single Program Multiple Data (SPMD) approach [6], where each of the M processes determines its assigned subset of $\mathbb{B}^N$ without coordination from a central process. Hence, the calculation can be transformed into an embarrassingly parallel problem [7], in which processes need not communicate except at the end of the computation, thereby allowing efficient scaling to many processes. Even with a large number of processes M, the number of paths 2^N quickly becomes exceedingly large as N increases. Therefore, we consider a Partitioned Monte Carlo method which uses a similar parallelization to reduce approximation error relative to basic Monte Carlo.

The rest of the paper is organized as follows. Section 2 introduces the binomial tree model to value an option using Bernoulli paths. Section 3 describes a parallel scheme to compute the expected value exactly. Section 4 presents the Partitioned Monte Carlo method to approximately compute the expected value. Section 5 presents results from the implementation of the methods for put options in R and Julia. Concluding remarks are given in Section 6.

2. Valuation of a path-dependent option using the binomial tree model

An option is a financial contract that gives the owner the right, but not the obligation, to either buy or sell a certain number of shares at a prespecified fixed price on a prespecified future date. A call option gives the owner the right to buy shares, while a put option gives the owner the right to sell shares. Several factors are used to value an option. The strike price K is a prespecified fixed price. The time T is the future date of maturity; for European options which are considered in this paper, the option can only be exercised at time T and subsequently becomes worthless. The value of an option is the amount a buyer is willing to pay when the option is bought. It depends on K, T, and the characteristics of the underlying stock. More formally, let $V(S_t)$ denote the value of the option at time t, at which time the price of the underlying stock is S_t. We assume that time starts at $t = 0$ at which point the option is bought or sold. The objective is to calculate $V(S_0)$, the value of the option at time $t = 0$. Although $V(S_t)$ for $t < T$ is not known, the value $V(S_T)$, called the payoff, is known with certainty. The value $V(S_T)$ of a call option at the time of maturity T is given by

$$V(S_T) = \max\{S_T - K, 0\}. \tag{2.1}$$

For a put option, the value at the time of maturity T is given by

$$V(S_T) = \max\{K - S_T, 0\}. \tag{2.2}$$

Note that in (2.1) and (2.2), the payoffs $V(S_T)$ depend only on the price of stock at time T, S_T, and the strike price K. In more complicated options, the payoffs often depend on additional factors. For example, the payoffs in path-dependent options depend on the historical price of the stock in a certain time period. For now, we will restrict our attention to simple options with payoffs in (2.1) and (2.2).

The binomial tree method of option valuation is based on simulating an evolution of the future price of the underlying stock between $t = 0$ and T using a recombinant binomial tree. We first discretize the interval $[0, T]$ into equidistant time steps. We select N to be the number

---
Algorithm 1 Build the grid of stock prices and calculate option payoffs for binomial method.
---
    for $i = 1, 2, \ldots, N$ **do**
        $S_{ij} = S_0 u^j d^{i-j}$ for $j = 0, 1, \ldots, i$
    end for
    for $j = 0, \ldots, N$ **do**
        $V_{Nj} \leftarrow \max\{S_{Nj} - K, 0\}$
    end for

---

of time steps, which determines the size of the tree, and let $\delta t = T/N$ be the size of each time step. Denote $t_i = i\,\delta t$ for $i = 0, \ldots, N$ as the distinct time points. Imagine a two-dimensional grid with t on the horizontal axis and stock price S_t on the vertical axis; by discretizing time, we slice the horizontal axis into equidistant time steps. We next discretize S_t at each $t = t_i$ resulting in values $S_{t_i j}$, where j is the index on the vertical axis. For notational convenience, we will write $S_{t_i j}$ as S_{ij}. The binomial tree method makes the following assumptions.

A1 The stock price S_{t_i} at t_i can only take two possible values over time step δt: price goes up to $S_{t_i} u$ or goes down to $S_{t_i} d$ at t_{i+1} with $0 < d < u$ where u is the factor of upward movement and d is the factor of downward movement. To enforce symmetry in the simulated stock prices, we assume $ud = 1$.

A2 The probability of moving up between time t_i and t_{i+1} is p for $i = 0, \ldots, N - 1$.

A3 $\mathrm{E}(S_{t_{i+1}} \mid S_{t_i}) = S_{t_i} e^{q \delta t}$, where q is the annual risk-free interest rate. For example, q may be the interest rate from a savings account at a high credit-worthy bank.

Under assumptions A1–A3, and if the stock price movements are assumed to be lognormally distributed with variance σ^2, it can be shown that

$$u = \beta + \sqrt{\beta^2 + 1},$$
$$p = (e^{q\delta t} - d)/(u - d),$$
$$\beta = \frac{1}{2}(e^{-q\delta t} + e^{(q+\sigma^2)\delta t}).$$

The standard deviation σ is also known as the volatility of the stock. For more details on deriving u and p, see Hull [2] or Seydel [3]. The above description follows the notations and development in Section 1.4 of Seydel [3] closely.

    Starting with the current stock price in the market S_0, a grid of possible future stock prices S_{ij} is built using u and p. Algorithm 1 shows the procedure to build a binomial tree of simulated future stock prices and calculate the payoffs at time T for a call option, for which, $V(S_T)$ is given by (2.1) at each j at time T. Therefore, $V_{Nj} = \max\{S_{Nj} - K, 0\}, j = 0, \ldots, N$, where V_{ij} is $V(S_{ij})$. Figure 2.1 shows a two-step recombinant binomial tree for a call option starting at the stock price S with the stock price evolution and option payoffs.

    In order to calculate the option value $V(S_0)$, the probabilities of reaching each of the leaf nodes of the tree must be calculated. These may be obtained from the probabilities of traversing each of the Bernoulli paths of dimension N. Since we assume that p is constant from A2, all the paths with the same number of up and down movements have the same probability of being traversed. The option value $V(S_0)$ is computed as the expectation of the payoffs discounted to the starting time $t = 0$ at the annual interest rate q as

$$V(S_0) = e^{-qT} \sum_{i=0}^{N} p(i) V_{Ni} = e^{-qT} \sum_{i=0}^{N} \binom{N}{i} p^i (1-p)^{N-i} V_{Ni}, \tag{2.3}$$
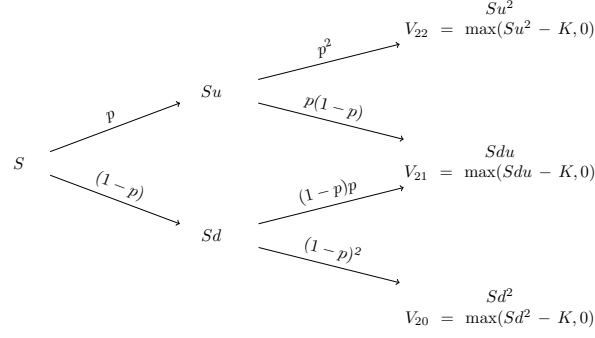
Figure 2.1.: A two-step recombinant binomial tree with call option payoffs.

where $p(i) = \binom{N}{i} p^i (1-p)^{N-i}$ is the probability of traversing paths ending at leaf node i, whose payoff is V_{Ni}.

Let $\boldsymbol{X} = (X_1, \ldots, X_N)$ represent a Bernoulli path where each $X_i \sim \text{Bernoulli}(p)$ independently for $i = 1, \ldots, N$. Figure 2.2 shows the two-step binomial tree in Figure 2.1 with Bernoulli paths to leaf nodes shown as vectors. The probability of taking path $\boldsymbol{x}$ is given by

$$\mathrm{P}(\boldsymbol{X} = \boldsymbol{x}) = p^{\boldsymbol{x}'\boldsymbol{1}}(1-p)^{N-\boldsymbol{x}'\boldsymbol{1}},$$

where $\boldsymbol{1}$ is an N-dimensional vector of ones. Since there are $\binom{N}{i}$ ways of reaching the leaf node i,

$$\mathrm{P}\{\text{reaching terminal node i}\} = \binom{N}{i} p^i (1-p)^{N-i}$$
$$= \sum_{\boldsymbol{x} \in \mathbb{B}^N : \boldsymbol{x}'\boldsymbol{1}=i} p^{\boldsymbol{x}'\boldsymbol{1}}(1-p)^{N-\boldsymbol{x}'\boldsymbol{1}}. \tag{2.4}$$

Substituting (2.4) in (2.3), we obtain

$$V(S_0) = e^{-qT} \sum_{i=0}^{N} V_{Ni} \sum_{\boldsymbol{x} \in \mathbb{B}^N : \boldsymbol{x}'\boldsymbol{1}=i} p^{\boldsymbol{x}'\boldsymbol{1}}(1-p)^{N-\boldsymbol{x}'\boldsymbol{1}}. \tag{2.5}$$

If the magnitudes and probabilities of up and down movements at each time step are constant, there is little computational advantage in evaluating the option value using (2.5) as opposed to (2.3). However, if the tree is built using time-varying up and down movements with corresponding probability p_t of an up movement at time t, or if the payoffs depend on the path $\boldsymbol{x}$, the model in (2.3) cannot be used. Let $p(\boldsymbol{x})$ be the probability of traversing the Bernoulli path $\boldsymbol{x}$ and $V_N(\boldsymbol{x})$ be the corresponding payoff. Since the space of Bernoulli paths is $\mathbb{B}^N$, (2.5) becomes

$$V(S_0) = e^{-qT} \sum_{i=0}^{N} \sum_{\boldsymbol{x} \in \mathbb{B}^N : \boldsymbol{x}'\boldsymbol{1}=i} p(\boldsymbol{x}) V_N(\boldsymbol{x}) = e^{-qT} \sum_{\boldsymbol{x} \in \mathbb{B}^N} V_N(\boldsymbol{x}) p(\boldsymbol{x}), \tag{2.6}$$

where $p(\boldsymbol{x}) = \prod_{i=1}^{N} p_i^{I(x_i=1)}(1-p_i)^{I(x_i=0)}$ and I is the indicator function. Note that (2.6) is similar to (1.2). We seek to parallelize the computation of the option value $V(S_0)$ in (2.6) or in general, the expected value in (1.2).
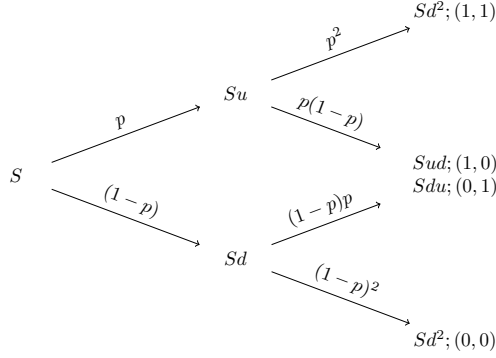
Figure 2.2.: A two-step binomial tree with Bernoulli paths.

3. Parallel Bernoulli Path Algorithm

Computation of the expected value (2.6) quickly becomes expensive as N increases, as 2^N Bernoulli paths must be considered. For example, taking $N = 24$ yields 16,777,216 possible paths. The computational burden can be efficiently shared by multiple processors by noting that the problem is embarrassingly parallel. Works such as Ganesan et al. [8] and Kolb and Pharr [9] have proposed parallel methods for evaluating option pricing models based on backward induction in a binary tree. To the best of our knowledge, the approach to parallelize the expected value computation using Bernoulli paths has not been considered before.

Our approach is based on the SPMD paradigm where a single program is executed on all the processes in parallel. It is not necessary for one process to preallocate the workload to individual processes; instead, each process can determine its share of the 2^N paths to work on. This is possible using the unique rank assigned to each process. Each process computes a local expected value on its partition of the sample space, and the final expected value is computed by summing across all processes. This summation is accomplished in the Message Passing Interface (MPI) framework through a *reduce* operation that coordinates communication between processes in an efficient way [6]. Utilizing the unique ranks of the processes to parallelize an algorithm is a common theme in parallel computing [6]. For example, Swarztrauber and Sweet [10] use the binary representation of the data to map computations to processes in a parallel direct solution of Poisson's equation. Here, we use the leading bits of the Bernoulli paths to map the paths onto processes.

Suppose there are M parallel processes with ranks $m = 0, \ldots, M - 1$; note that ranks traditionally start at 0 in the MPI framework. The process with rank m will be referred to as "process m". We assume that $M \leq N$ and that M is a power of 2. Let $r = \log_2(M)$ so that the rank m of a process can be written with the r-digit binary representation $m = z_{r-1}2^{r-1} + \cdots + z_1 2^1 + z_0 2^0$, where each $z_j \in \mathbb{B}$. Process m is assigned all paths $\boldsymbol{x}$ with prefix $(z_{r-1}, \ldots, z_1, z_0)$; this set of 2^{N-r} paths is denoted

$$\mathbb{B}_m^N = \{\boldsymbol{x} \in \mathbb{B}^N : x_1 = z_{r-1}, \ldots, x_{r-1} = z_1, x_r = z_0\}.$$

Note that the sets $\mathbb{B}_m^N$ form a partition of $\mathbb{B}^N$. Figure 3.1 shows a diagram of the mapping from m to $\mathbb{B}_m^N$. For each Bernoulli path $\boldsymbol{x} \in \mathbb{B}_m^N$, process m computes the probability of traversing the path $p(\boldsymbol{x})$ and the payoff value $V_N(\boldsymbol{x})$. The local expected value of the payoffs V_m on process m is calculated as

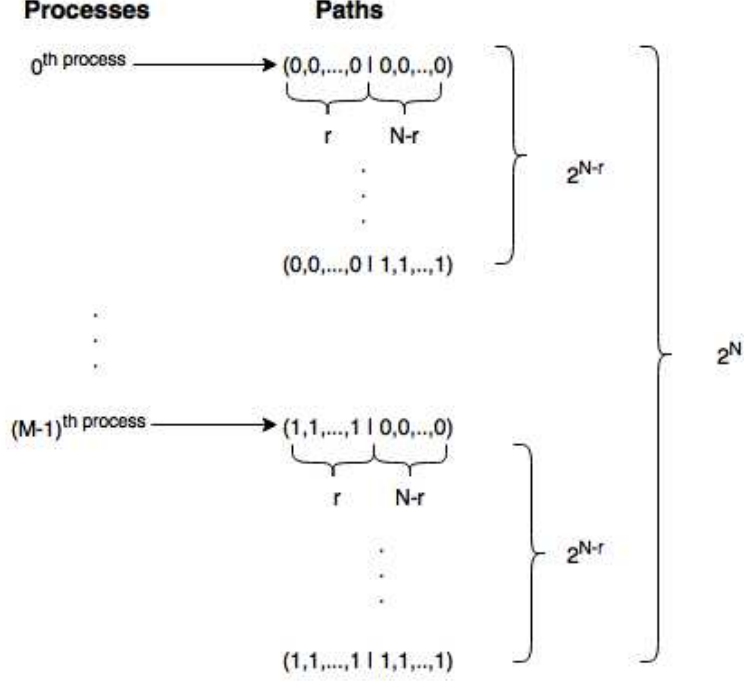$$V_m = e^{-qT} \sum_{\boldsymbol{x} \in \mathbb{B}_m^N} p(\boldsymbol{x}) V_N(\boldsymbol{x}). \tag{3.1}$$

Figure 3.1.: Process-Bernoulli path mapping

Finally, local expected values are summed to produce the final result

$$V(S_0) = \sum_{m=0}^{M-1} V_m; \tag{3.2}$$

this is implemented by an MPI reduce operation to obtain the result on process 0. The computation of (3.2) requires 2^{N-r} steps on 2^r parallel processes rather than 2^N steps on a single process, as required in the serial computation. The method can be extended to the case when M is not exactly a power of 2 if we are willing to forfeit perfect load balancing. For example, we can consider a partition $\mathbb{B}^N = \mathbb{B}_0^N \cup \cdots \cup \mathbb{B}_K^N$ for some $K >> M$. Process 0 can handle $\mathbb{B}_m^N$ for $m = 0, M, 2M, \ldots$, process 1 can handle $\mathbb{B}_m^N$ for $m = 1, M+1, 2M+1, \ldots$, and so forth. Note that the idea of partitioning the paths is applicable to trees with more than two branches at each node. Examples of such trees include trinomial trees[11].

4. Monte Carlo Estimation and Variance Reduction

Recall that the number of Bernoulli paths in the set $\mathbb{B}^N$ grows exponentially with N. When N becomes large, it is infeasible to compute the expected value (2.6) exactly, even with a reasonably large number of processors. Monte Carlo (MC) estimation provides a way to approximate a complicated expected value without enumerating the entire sample space. In this section, we discuss an MC method that uses the partitioning scheme from Section 3 to approximate the result using M parallel processes. The mth process is given the responsibility of drawing from $\mathbb{B}_m^N$, for $m = 0, \ldots, M-1$, so that we effectively enumerate the first $r = \log_2 M$ steps of each path, and draw the rest through Monte Carlo. This provides a reduction in variance over a basic MC estimator that uses the same number of draws.

Define

$$\theta = \mathrm{E}[V(\boldsymbol{X})] = \sum_{\boldsymbol{x} \in \mathbb{B}^N} V(\boldsymbol{x}) p(\boldsymbol{x}),$$

where the suffix N in $V(\boldsymbol{x})$ is dropped for notational convenience. The option value in (2.6) can be written as $V = e^{-qT} \theta$. Given an estimate $\hat{\theta}$ of θ, an estimate of V is $\hat{V} = e^{-qT} \hat{\theta}$, its variance is $\mathrm{Var}(\hat{V}) = e^{-2qT} \mathrm{Var}(\hat{\theta})$, and an estimate of the variance is $\widehat{\mathrm{Var}}(\hat{V}) = e^{-2qT} \widehat{\mathrm{Var}}(\hat{\theta})$. Therefore, we will focus on estimating θ for the remainder of this section.

Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_R$ be R independent and identically distributed (i.i.d.) Bernoulli paths sampled from $\mathbb{B}^N$. Then the MC estimator of θ is given by

$$\hat{\theta} = \frac{1}{R} \sum_{i=1}^{R} V(\boldsymbol{x}_i)$$

and its variance is

$$\mathrm{Var}(\hat{\theta}) = \frac{1}{R} \mathrm{Var}[V(\boldsymbol{X})], \tag{4.1}$$

which can be estimated from the MC draws by

$$\widehat{\mathrm{Var}}(\hat{\theta}) = \frac{1}{R^2} \sum_{i=1}^{R} \left(V(\boldsymbol{x}_i) - \hat{\theta} \right)^2.$$

In Section 3, we partitioned the space $\mathbb{B}^N$ of Bernoulli paths into $\mathbb{B}_0^N, \ldots, \mathbb{B}_M^N$. Let $\mathcal{D}_m$ denote the event $[\boldsymbol{X} \in \mathbb{B}_m^N]$ which occurs with probability $\mathrm{P}(\mathcal{D}_m)$, for $m = 0, \ldots, M-1$. Furthermore, consider the partitioning $\boldsymbol{X} = (\boldsymbol{Z}, \boldsymbol{Y})$ where $\boldsymbol{Z} \in \mathbb{B}^r$ and $\boldsymbol{Y} \in \mathbb{B}^{N-r}$. We can now write θ as

$$\theta = \sum_{m=0}^{M-1} \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}_m] \, \mathrm{P}(\mathcal{D}_m), \tag{4.2}$$

where

$$\mathrm{P}(\mathcal{D}_m) = \sum_{\boldsymbol{y} \in \mathbb{B}^{N-r}} \mathrm{P}(\boldsymbol{Z} = \boldsymbol{z}_m, \boldsymbol{Y} = \boldsymbol{y}) = \mathrm{P}(\boldsymbol{Z} = \boldsymbol{z}_m)$$

and $\boldsymbol{z}_m$ is the binary representation of m corresponding to the rank of the mth process. Let $\theta^{(m)} = \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}_m]$ and let $\boldsymbol{x}_1^{(m)}, \ldots, \boldsymbol{x}_{R_m}^{(m)}$ be an i.i.d. sample from the distribution of paths on $\mathbb{B}_m^N$ for each $m = 0, \ldots, M-1$. Suppose $\sum_{m=0}^{M-1} R_m = R$ so that the sample size used is as in the basic MC estimator. The estimator

$$\hat{\theta}_s^{(m)} = \frac{1}{R_m} \sum_{i=1}^{R_m} V(\boldsymbol{x}_i^{(m)})$$

is an unbiased estimator of $\theta^{(m)}$ with variance $\frac{1}{R_m} \mathrm{Var}[V(\boldsymbol{X}) \mid \mathcal{D}_m]$. Substituting $\hat{\theta}_s^{(m)}$ for $\theta^{(m)}$

in (4.2) yields the Partitioned MC estimator

$$\hat{\theta}_s = \sum_{m=0}^{M-1} \hat{\theta}_s^{(m)} \, \mathrm{P}(\mathcal{D}_m). \tag{4.3}$$

Following Rubinstein and Kroese [12], we choose sample sizes R_m proportional to $\mathrm{P}(\mathcal{D}_m)$ as $R_m = R \cdot \mathrm{P}(\mathcal{D}_m)$ for each m. With this choice, and ignoring that $R \cdot \mathrm{P}(\mathcal{D}_m)$ likely will not be an exact integer, the variance of the Partitioned MC estimator can be written

$$\begin{aligned}
\mathrm{Var}(\hat{\theta}_s) &= \sum_{m=0}^{M-1} \mathrm{Var}(\hat{\theta}^{(m)})[\mathrm{P}(\mathcal{D}_m)]^2 \\
&= \frac{1}{R} \sum_{m=0}^{M-1} \mathrm{Var}[V(\boldsymbol{X}) \mid \mathcal{D}_m] \, \mathrm{P}(\mathcal{D}_m).
\end{aligned} \tag{4.4}$$

A corresponding variance estimator is

$$\begin{aligned}
\widehat{\mathrm{Var}}(\hat{\theta}_s) &= \frac{1}{R} \sum_{m=0}^{M-1} \left[\frac{1}{R_m} \sum_{i=1}^{R_m} \left(V(\boldsymbol{x}_i^{(m)}) - \hat{\theta}_m \right)^2 \right] \mathrm{P}(\mathcal{D}_m) \\
&= \frac{1}{R^2} \sum_{m=0}^{M-1} \sum_{i=1}^{R_m} \left(V(\boldsymbol{x}_i^{(m)}) - \hat{\theta}_m \right)^2.
\end{aligned}$$

To verify that $\hat{\theta}_s$ gives a variance reduction over $\hat{\theta}$, the law of total variation gives

$$\begin{aligned}
\mathrm{Var}[V(\boldsymbol{X})] &= \mathrm{E}_{\mathcal{D}} \, \mathrm{Var}[V(\boldsymbol{X}) \mid \mathcal{D}] + \mathrm{Var}_{\mathcal{D}} \, \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}] \tag{4.5} \\
&= \sum_{m=0}^{M-1} \mathrm{Var}[V(\boldsymbol{X}) \mid \mathcal{D}_m] \, \mathrm{P}(\mathcal{D}_m) + \mathrm{Var}_{\mathcal{D}} \, \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}] \\
&= R \, \mathrm{Var}(\hat{\theta}_s) + \mathrm{Var}_{\mathcal{D}} \, \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}], \tag{4.6}
\end{aligned}$$

where the last equality is from (4.4). Substituting the left hand side in (4.6) in terms of $\mathrm{Var}(\hat{\theta})$ from (4.1) and dividing both sides by R we get

$$\mathrm{Var}(\hat{\theta}) = \mathrm{Var}(\hat{\theta}_s) + \frac{\mathrm{Var}_{\mathcal{D}} \, \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}]}{R}. \tag{4.7}$$

Note that $\mathrm{Var}_{\mathcal{D}} \, \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}] = 0$ if $V(\boldsymbol{X}) \mid \mathcal{D}$ does not depend on the first r steps of the Bernoulli paths or when $r \in \{0, N\}$, that is, when the number of processes $M \in \{1, 2^N\}$. When $M = 1$, the Partitioned MC method is same as the basic MC method and when $M = N$, it is same as the exact expected value in (2.6). Since the payoff $V(\boldsymbol{X})$ is assumed to depend on the entire path $\boldsymbol{X}$, the second term in the right hand side of (4.7) is greater than 0 when $0 < r < N$ and, therefore, the Partitioned MC estimator in (4.3) typically yields strict reduction in the variance. The reduction will be more pronounced when $V(\boldsymbol{X})$ are heterogeneous across the $\mathcal{D}_m$ and homogeneous within each $\mathcal{D}_m$.

Remark 1. An interesting variation of the Partitioned MC method is to reuse the same sample of R draws from $\mathbb{B}^{N-r}$ on all processes. Consider again the partitioning $\boldsymbol{X} = (\boldsymbol{Z}, \boldsymbol{Y})$, and suppose $\boldsymbol{Z}$ and $\boldsymbol{Y}$ are independent. Let $\boldsymbol{x}_i^{(m)} = (\boldsymbol{z}_m, \boldsymbol{y}_i)$, where $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_R$ are i.i.d. draws

from the distribution of $\boldsymbol{Y}$ and $\boldsymbol{z}^{(m)} = (z_{r-1}^{(m)}, \ldots, z_1^{(m)}, z_0^{(m)})$ is the binary representation of m corresponding to the rank of the mth process. Then an estimate for $\theta^{(m)}$ is given by

$$\tilde{\theta}^{(m)} = \frac{1}{R} \sum_{i=1}^{R} V(\boldsymbol{x}_i^{(m)}) \tag{4.8}$$

and θ in (4.2) can be estimated unbiasedly by

$$\begin{aligned}
\tilde{\theta} &= \sum_{m=0}^{M-1} \tilde{\theta}^{(m)} \, \mathrm{P}(\mathcal{D}_m) \\
&= \frac{1}{R} \sum_{i=1}^{R} \sum_{m=0}^{M-1} V(\boldsymbol{x}_i^{(m)}) \, \mathrm{P}(\mathcal{D}_m) \\
&= \frac{1}{R} \sum_{i=1}^{R} \mathrm{E}_{\boldsymbol{X}|\boldsymbol{Y}}[V(\boldsymbol{Z}, \boldsymbol{y}_i)]. \tag{4.9}
\end{aligned}$$

The variance of this estimator is $\mathrm{Var}(\tilde{\theta}) = \frac{1}{R} \mathrm{Var}_{\boldsymbol{Y}} \mathrm{E}_{\boldsymbol{X}|\boldsymbol{Y}}[V(\boldsymbol{X})]$ and an estimate of the variance from the MC sample is

$$\begin{aligned}
\widehat{\mathrm{Var}}(\tilde{\theta}) &= \frac{1}{R^2} \sum_{i=1}^{R} \left(\mathrm{E}_{\boldsymbol{X}|\boldsymbol{Y}}[V(\boldsymbol{Z}, \boldsymbol{y}_i)] - \tilde{\theta}^{(m)} \right)^2 \\
&= \frac{1}{R^2} \sum_{i=1}^{R} \left(\sum_{m=0}^{M-1} V(\boldsymbol{x}_i^{(m)}) \, \mathrm{P}(\mathcal{D}_m) - \tilde{\theta}^{(m)} \right)^2.
\end{aligned}$$

We refer to $\tilde{\theta}$ as the Shared Sample MC estimator. Now, the variance of $V(\boldsymbol{X})$ can be written as

$$\begin{aligned}
\mathrm{Var}[V(\boldsymbol{X})] &= \mathrm{Var}_{\boldsymbol{Y}} \mathrm{E}[V(\boldsymbol{X}) \mid \boldsymbol{Y}] + \mathrm{E}_{\boldsymbol{Y}} \mathrm{Var}[V(\boldsymbol{X}) \mid \boldsymbol{Y}] \\
&= R \, \mathrm{Var}(\tilde{\theta}) + \mathrm{E}_{\boldsymbol{Y}} \mathrm{Var}[V(\boldsymbol{X}) \mid \boldsymbol{Y}]. \tag{4.10}
\end{aligned}$$

Substituting the left hand side in (4.10) in terms of $\mathrm{Var}(\hat{\theta})$ from (4.1) and dividing both sides by R we get

$$\mathrm{Var}(\hat{\theta}) = \mathrm{Var}(\tilde{\theta}) + \frac{\mathrm{E}_{\boldsymbol{Y}} \mathrm{Var}[V(\boldsymbol{X}) \mid \boldsymbol{Y}]}{R}. \tag{4.11}$$

Again, the second term in the right hand side of the (4.11) is zero if and only if $V(\boldsymbol{X}) \mid \boldsymbol{Y}$ does not depend on the first r steps of the Bernoulli paths, for $0 < r < N$. Since we assume $V(\boldsymbol{X})$ depends on the entire path $\boldsymbol{X}$, $\mathrm{Var}(\tilde{\theta})$ is strictly less than $\mathrm{Var}(\hat{\theta})$. The following result summarizes the relationship among the variances of $\hat{\theta}$, $\hat{\theta}_s$, and $\tilde{\theta}$.

Theorem 4.1. *Suppose $R_m = R$ for $m = 0, \ldots, M-1$, $0 < r < N$, and $V(\boldsymbol{Z}_k, \boldsymbol{Y})$ and $V(\boldsymbol{Z}_l, \boldsymbol{Y})$ are positively correlated for all $k, l \in \{0, \ldots, M-1\}$. Then $\mathrm{Var}(\hat{\theta}_s) \leq \mathrm{Var}(\tilde{\theta}) \leq \mathrm{Var}(\hat{\theta})$.*

Proof. We have already shown that $\mathrm{Var}(\tilde{\theta}) \leq \mathrm{Var}(\hat{\theta})$. Now we have $\mathrm{Var}(\hat{\theta}_s) =$

$\frac{1}{R}\sum_{m=0}^{M-1} \mathrm{Var}[V(\boldsymbol{X}) \mid \mathcal{D}_m][\mathrm{P}(\mathcal{D}_m)]^2$, and from (4.9),

$$
\begin{aligned}
\mathrm{Var}(\tilde{\theta}) &= \frac{1}{R}\,\mathrm{Var}_{\boldsymbol{Y}}\left[\sum_{m=0}^{M-1} V(\boldsymbol{Z}_m, \boldsymbol{Y})\,\mathrm{P}(\mathcal{D}_m)\right] \\
&= \frac{1}{R}\sum_{m=0}^{M-1}\mathrm{Var}_{\boldsymbol{Y}}[V(\boldsymbol{Z}_m, \boldsymbol{Y})][\mathrm{P}(\mathcal{D}_m)]^2 + \frac{1}{R}\sum_{k\neq l}\sum \mathrm{Cov}\left(V(\boldsymbol{Z}_k, \boldsymbol{Y}), V(\boldsymbol{Z}_l, \boldsymbol{Y})\right)\mathrm{P}(\mathcal{D}_k)\,\mathrm{P}(\mathcal{D}_l) \\
&= \mathrm{Var}(\hat{\theta}_s) + \frac{1}{R}\sum_{k\neq l}\sum \mathrm{Cov}\left(V(\boldsymbol{Z}_k, \boldsymbol{Y}), V(\boldsymbol{Z}_l, \boldsymbol{Y})\right)\mathrm{P}(\mathcal{D}_k)\,\mathrm{P}(\mathcal{D}_l) \\
&\geq \mathrm{Var}(\hat{\theta}_s).
\end{aligned}
$$

$\square$

5. Application to option pricing

We implemented the method described in Section 3 to value a put option using the R 3.2.2 and Julia 0.4.6 programming environments. Computations were run on a distributed cluster with compute nodes, each having two Intel E5-2650v2 Ivy Bridge (2.6 GHz, 20 MB cache) processors with 8 cores per node, for a total of 16 cores per node. All nodes have 64 GB of main memory and are connected by a quad-data rate InfiniBand interconnect. Open MPI 1.8.5 [13] was used as the underlying implementation of the MPI framework.

R is a statistical computing environment that facilitates advanced data analysis, provides graphical capabilities and an interpreted high-level programming language [14]. On top of the statistical, computational, and programmatic features available in the core R environment, additional capabilities are available through numerous packages which have been contributed by the user community. The Rmpi [15] and pbdMPI [16] packages may be used to write MPI programs from R. Results shown in this section are based on Rmpi, but pbdMPI performed similarly in our experience. The package Rcpp [17] facilitates integration of C++ code into R programs, which can substantially improve performance at the cost of an increased programming burden. We have not yet explored Rcpp in our implementation, but note its potential use.

Julia is a recently developed programming language that is gaining popularity in scientific computing, data analysis, and high performance computing [18]. It is a compiled language that uses the Low Level Virtual Machine Just-in-Time technology [19] to generate an optimized version of the source code compiled to the machine level. Julia provides a number of computational and statistical capabilities, both in the core environment and through packages contributed by the user community. We have used the package MPI [20] to run MPI programs in Julia. Integration with C++ is also possible in Julia through packages such as CxxWrap and Cpp, but we have not yet explored their use. Our implementation uses native Julia code with the MPI package. Because Julia is compiled into machine-level code, it is expected that a program written in Julia will perform better than an equivalent program written in R. Performance results later in this section confirm our hunch.

Listing 1 shows a snippet of our Julia implementation of the parallelization method. Since the structure of our R and Julia implementations are similar, we do not show a similar listing of our R code. In line 1 we load the MPI package. Since our implementation follows the SPMD paradigm, the same code runs on all the processes. The rank of the process on which the code is being run is requested on line 5 and on line 6 the total number of processes in the MPI communicator is requested. As the while loop at line 11 shows, each process works on 2^{N-r} out of the total 2^N Bernoulli paths. Note the construction of the full Bernoulli path in line 13 by prepending the binary representation of the rank of the specific process on which the code is being run to

the current $(N-r)$-dimensional Bernoulli path. The function call to `calc_path_prob` on line 14 calculates the probability of traversing the Bernoulli path constructed in line 13. The function call to `calc_payoff` on line 15 calculates the option payoff; their function definitions are not shown because they are independent of the parallelization method. Finally, on line 21, expected values from individual processes are summed together to obtain the final answer at process 0.

```
1  import MPI
2  .
3  MPI.Init()
4  comm = MPI.COMM_WORLD
5  id = MPI.Comm_rank(comm)
6  M = MPI.Comm_size(comm)
7  .
8  r = log2(M)
9  l_n = convert(Int64, 2^(N-r))
10 .
11 while i < l_n
12     node = i
13     path = cat(2, integer_base_b(id, 2, r), integer_base_b(node, 2, N-r))
14     p_vt = calc_path_prob(path, probs)
15     vt = calc_payoff(S, K, u, d, opt_type, path)
16     v += p_vt*vt
17     i += 1
18 end
19 .
20 v = exp(-q*T)*v
21 reduced_v = MPI.Reduce(v, MPI.SUM, 0, comm)
```

Listing 1: A `Julia` implementation of the parallel Bernoulli path algorithm.

We take a put option as an example to illustrate our methodology. We set a strike price of $K = 10$. Current price and volatility of the asset are $S = 5$ and $\sigma = 0.30$, respectively. Risk-free interest rate is $q = 6\%$ and time to maturity T is one year. Tables 5.1(a) and 5.2(a) show the wall clock runtimes of our `R` and `Julia` implementations, respectively, for problem sizes $N = 16, 20, 24, 28$, and 32. While both the implementations scale well with the number of processes M, the `Julia` implementation is roughly 10 times faster than `R`. Our `R` program for $N = 32$ on a single process ($M = 1$) resulted in an overflow in the loop that computes the expected value since $2^{31}-1$ is the maximum integer value that can be stored in `R`. As a result, the runtime for this particular case is recorded as N/A in Table 5.1(a). If T_M is the runtime taken for M number of processes, the speedup S_M and efficiency E_M for M are defined as T_1/T_M and S_M/M respectively. If the program scales up perfectly to M processes, ideal values $S_M = M$ and $E_M = 1$ are obtained. These numbers indicate the scalability of the program. Since our `R` program did not run on a single process for $N = 32$, we take the speedup for this case to be $2 \cdot T_2/T_M$, $M = 2, \ldots, 64$ and for $M = 1$ and $M = 2$, the speedups are taken to be 1 and 2 respectively. Tables 5.1(b) and 5.2(b) show the speedups and Tables 5.1(c) and 5.2(c) show the efficiency numbers of our `R` and `Julia` implementations, respectively. The plots in Figure 5.1 visualize the speedup and efficiency numbers in Tables 5.1(b) and 5.1(c), respectively, and Figure 5.2 shows the corresponding plots from Table 5.2. These plots visually confirm our conjecture that `Julia` is more efficient than `R` for our problem. Note that for a fixed problem size, there is a reduced advantage in the speedup beyond a certain number of tasks. This is because the overhead of coordinating the tasks begins to dominate the time spent doing useful calculations; see Pacheco [6] for more details. This can be seen for $N = 16$ in Table 5.1(a) and for $N = 16$ and $N = 20$ in Table 5.2(a). In both the cases, since the very small total runtime is mostly dominated by the near constant time consumed by the MPI reduce operation, the speedup and efficiency numbers are significantly lower than larger sized problems. The relatively high

speedup and efficiency numbers for $N = 32$ in Tables 5.1(b) and 5.1(c) resulted from fixing the speedup for $M = 2$ to be 2 and calculating the rest of the speedup and efficiency numbers relative to $M = 2$. Since the runtimes for `Julia` are roughly 10 times faster than those for `R`, if we estimate the runtime for $N = 32$ and $M = 1$ for `R` and accordingly calculate the speedup and efficiency numbers, we will notice that the numbers drop and are comparable to rest of the cases.

We implemented the Monte Carlo estimation methods described in Section 4 for Asian and Lookback options, which are both path-dependent [2]. In an Asian option, the asset price S_T at the time of maturity is replaced in the option payoff function with the arithmetic average of $\{S_t : t = 1, \ldots, N\}$. Therefore, in the binomial tree model, the payoff for an Asian put option is given by

$$V(\boldsymbol{x}) = \max\{K - S^*, 0\}, \tag{5.1}$$

where $S^* = \frac{1}{N} \sum_{t=1}^{N} S_t(\boldsymbol{x})$, $S_t(\boldsymbol{x})$ is the asset value at time t followed on the Bernoulli path $\boldsymbol{x}$. In a Lookback option, either the strike price K or the asset price S_T at the time of maturity are replaced in the payoff function by the maximum or minimum of $\{S_t\}$ respectively. Here we consider a Fixed Lookback put option, whose payoff is given by

$$V(\boldsymbol{x}) = \max\{K - S^*, 0\},$$

where $S^* = \min\{S_t(\boldsymbol{x}) : t = 1, \ldots, N\}$. We implemented the basic MC estimate given in section 4 for the Asian and Fixed Lookback put options using the binomial tree model with size N to study the convergence of the estimates to the exact expected value (2.6). We further implemented Partitioned and Shared Sample MC from section 4 to study the variance reduction property. Table 5.3 shows basic MC estimates and corresponding variance estimates of an Asian put option and a Fixed Lookback put option with parameters $K = 100$, $S = 20$, $q = 6\%$, $\sigma = 3.0$, and $T = 1$, using the binomial model with tree size $N = 32$. Option values calculated by exact enumeration were 82.115 for the Asian put options and 93.196 for the Fixed Lookback put option. The sample size used for the MC estimation is increased from 2^9 to 2^{16}, which is less than 0.01% of the total number of paths. As can be seen from Table 5.3, MC estimates for both the options converge to their respective exact values. Also, as expected, the variance estimates decrease with increasing sample size R. Table 5.4 shows the Partitioned MC estimates $\hat{V}_s$ for both the Asian and Fixed Lookback put options and corresponding variance estimates, using a total sample size of $R = 1024$ and varying the number of processes between 1 to 64. Note that as the number of processes increase, the sample size per process R_m decreases. The estimates shown in Table 5.4 are averaged over 1000 repetitions. As expected, Table 5.3 shows that variance estimates of the Partitioned MC estimator are mostly smaller than the corresponding basic MC estimator for $R = 1024$. Table 5.5 shows the comparison of variance estimates between the Partitioned and Shared Sample MC estimates for the Asian put option with $N = 32$, $R = R_m = 1024$, and $m = 0, \ldots, M - 1$. Shared Sample MC estimates $\tilde{V}$ and corresponding variance estimates were calculated using the expressions given in section 4. Again, the estimates in Table 5.5 are averaged over 1000 repetitions. The results show that if $R = R_m$, and $m = 0, \ldots, M - 1$, the Partitioned MC method reduces the variance of the estimator more than the Shared Sample method does, as expected from Theorem 4.1. The condition on the covariance between $V(\boldsymbol{z}_k, \boldsymbol{y})$ and $V(\boldsymbol{z}_l, \boldsymbol{y})$ for all $k, l \in \{0, \ldots, M - 1\}$ where $k \neq l$ is satisfied for the options considered here.

Table 5.1.: Runtime for different number of time steps for `R` implementation. For $M = 1$, $N = 32$, since our program failed to run because of integer overflow, runtime is shown as N/A.

(a) Wall clock time in HH:MM:SS						
N	$M = 1$	2	4	8	16	32	64
16	00:00:04	00:00:02	00:00:01	<00:00:01	<00:00:01	<00:00:01	<00:00:01
20	00:01:09	00:00:43	00:00:23	00:00:12	00:00:06	00:00:03	00:00:02
24	00:20:22	00:12:41	00:06:56	00:03:35	00:01:47	00:00:54	00:00:27
28	06:00:09	03:41:26	01:59:31	01:02:16	00:31:18	00:16:02	00:08:00
32	N/A	65:54:36	35:18:59	18:41:10	10:38:06	04:44:09	02:22:59

(b) Observed speedup S_M						
N	$M = 1$	2	4	8	16	32	64
16	1.00	1.60	2.97	5.55	10.88	17.69	27.11
20	1.00	1.59	2.98	5.72	11.62	22.86	44.33
24	1.00	1.60	2.93	5.68	11.38	22.62	44.32
28	1.00	1.62	3.01	5.78	11.50	22.45	45.01
32	N/A	2.00	3.73	7.05	13.99	28.21	55.31

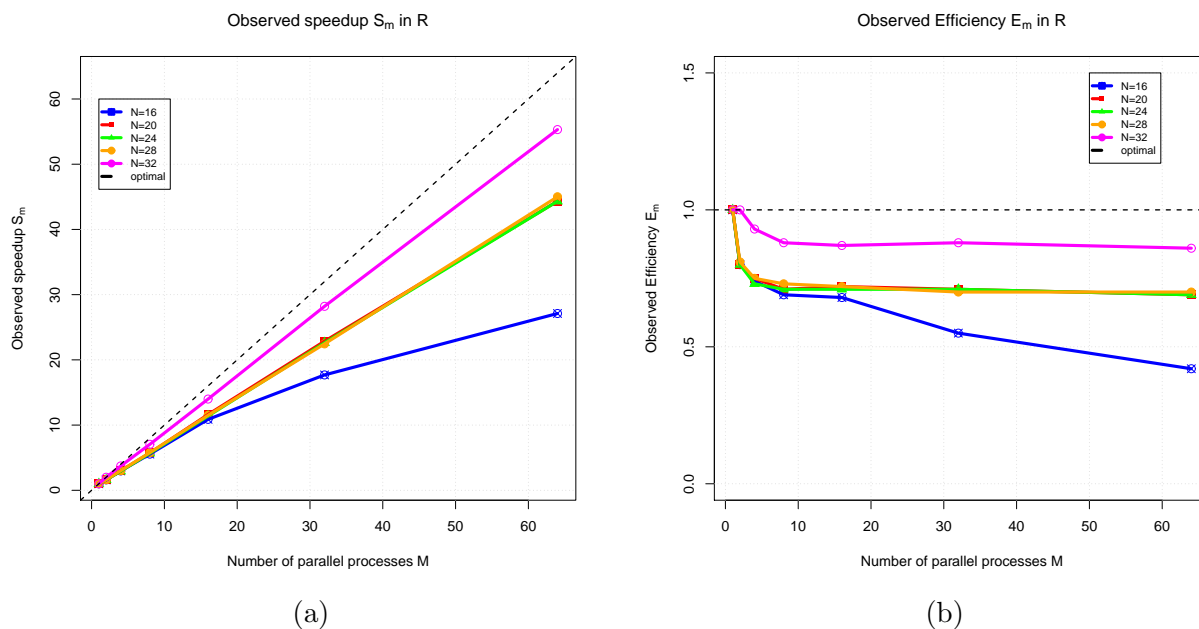| (c) Observed efficiency E_M						
N	$M = 1$	2	4	8	16	32	64
16	1.00	0.80	0.74	0.69	0.68	0.55	0.42
20	1.00	0.80	0.75	0.71	0.72	0.71	0.69
24	1.00	0.80	0.73	0.71	0.71	0.71	0.69
28	1.00	0.81	0.75	0.73	0.72	0.70	0.70
32	N/A	1.00	0.93	0.88	0.87	0.88	0.86

Figure 5.1.: (a) Speedup and (b) Efficiency in `R`.

Table 5.2.: Runtime for different number of time steps for `Julia` implementation.

(a) Wall clock time in HH:MM:SS

N	$M = 1$	2	4	8	16	32	64
16	<00:00:01	<00:00:01	<00:00:01	<00:00:01	<00:00:01	<00:00:01	<00:00:01
20	00:00:07	00:00:05	00:00:02	<00:00:01	<00:00:01	<00:00:01	<00:00:01
24	00:02:23	00:01:15	00:00:42	00:00:22	00:00:11	00:00:06	00:00:03
28	00:40:08	00:21:05	00:11:02	00:05:56	00:03:07	00:01:34	00:00:51
32	11:59:54	06:23:01	03:24:58	01:46:06	00:53:41	00:27:18	00:13:38

(b) Observed speedup S_M

N	$M = 1$	2	4	8	16	32	64
16	1.00	1.57	2.19	2.74	3.36	4.36	4.04
20	1.00	1.80	3.16	6.18	11.10	17.65	25.05
24	1.00	1.91	3.36	6.37	13.02	24.76	47.79
28	1.00	1.90	3.64	6.76	12.90	25.71	47.06
32	1.00	1.88	3.52	6.76	13.44	26.37	52.58

(c) Observed efficiency E_M

N	$M = 1$	2	4	8	16	32	64
16	1.00	0.78	0.55	0.34	0.21	0.13	0.06
20	1.00	0.90	0.79	0.77	0.69	0.55	0.39
24	1.00	0.95	0.84	0.79	0.81	0.77	0.75
28	1.00	0.95	0.91	0.86	0.81	0.80	0.73
32	1.00	0.94	0.88	0.85	0.84	0.82	0.82
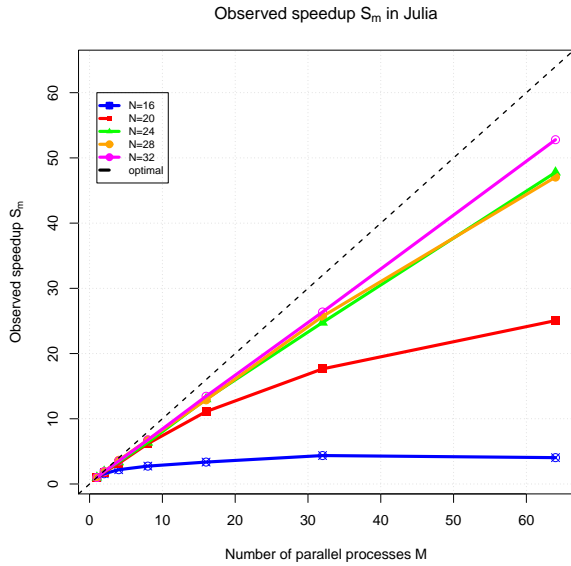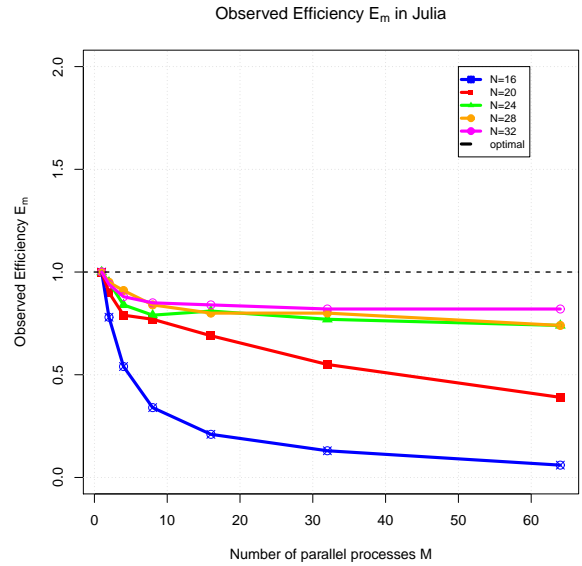
Figure 5.2.: (a) Speedup and (b) Efficiency in `Julia`.

Table 5.3.: Monte Carlo estimates and corresponding variance estimates for Asian and Fixed Lookback put options with $N = 32$. Exact value of the Asian option is 82.115 and the Lookback option is 93.196.

Option	Estimate	$R = 2^9$	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
Asian	$\hat{V}$	82.857	82.514	83.425	83.181	82.821	82.615	82.566	82.524
	$\widehat{\text{Var}}(\hat{V})$	0.735	0.362	0.179	0.095	0.050	0.022	0.011	0.006
Lookback	$\hat{V}$	93.156	93.237	93.312	93.324	93.262	93.236	93.234	93.222
	$\widehat{\text{Var}}(\hat{V})$	0.022	0.009	0.005	0.002	0.001	<0.001	<0.001	<0.001

Table 5.4.: Partitioned Monte Carlo estimates and corresponding variance estimates for Asian and Fixed Lookback put options with $N = 32$ and $R = 2^{10}$.

Option	Estimate	$M = 1$ $R_m = 2^{10}$	2 2^9	4 2^8	8 2^7	16 2^6	32 2^5	64 2^4
Asian	$\hat{V}_s$	82.077	82.217	82.101	82.232	81.936	82.296	82.165
	$\widehat{\text{Var}}(\hat{V}_s)$	0.367	0.332	0.315	0.272	0.263	0.212	0.194
Lookback	$\hat{V}_s$	93.196	93.201	93.171	93.197	93.187	93.216	93.205
	$\widehat{\text{Var}}(\hat{V}_s)$	0.010	0.009	0.008	0.008	0.008	0.007	0.007

Table 5.5.: Comparison of the variance estimates from the Partitioned and Shared Sample Monte Carlo methods with $N = 32$ and $R = 1024$ for Asian put option.

Method	Estimate	$M = 1$	2	4	8	16	32	64
Partitioned MC	$\hat{V}_s$	82.201	82.160	82.101	82.127	82.109	82.120	82.108
	$\widehat{\text{Var}}(\hat{V}_s)$	0.367	0.220	0.062	0.027	0.005	0.005	0.002
Shared Sample MC	$\tilde{V}$	82.113	82.112	82.120	82.093	82.102	82.124	82.108
	$\widehat{\text{Var}}(\tilde{V})$	0.373	0.305	0.267	0.203	0.170	0.142	0.123

6.   Concluding Remarks

We have presented a method to transform the computation of the expected value in a recombinant binomial tree into an embarrassingly parallel problem by mapping the Bernoulli paths in the tree to the processes on a multiprocessor computer. We also discussed a parallel Monte Carlo estimation method which takes advantage of this partitioning. The methods were implemented both in `R` and `Julia`, and were applied to value path-dependent options. Numerical results verify the convergence of the parallel Monte Carlo method and variance reduction with respect to basic Monte Carlo estimation. Performance results indicate that the `Julia` implementation was significantly faster and more efficient than the `R` implementation, likely because of the superior handling of loops and the compilation to machine-level code.

References

[1] Cox JC, Ross SA, Rubinstein M. Option pricing: A simplified approach. Journal of Financial Economics. 1979;7(3):229–263. Available from: `http://www.sciencedirect.com/science/article/pii/0304405X79900151`.

[2] Hull JC. Options, futures, and other derivatives. Prentice Hall; 2003.

[3] Seydel R. Tools for computational finance. Springer; 2003.

[4] Little RJA. Pattern-mixture models for multivariate incomplete data. Journal of the American Statistical Association. 1993;88(421):125–134.

[5] Glasserman P. Monte carlo methods in financial engineering. 1st ed. Springer-Verlag; 2003.

[6] Pacheco PS. Parallel programming with mpi. Morgan Kaufmann; 1997.

[7] Foster I. Designing and building parallel programs: Concepts and tools for parallel software engineering. Addison-Wesley; 1995.

[8] Ganesan N, Chamberlain RD, Buhler J. Acceleration of binomial options pricing via parallelizing along time-axis on a GPU. Proc of Symp on Application Accelerators in High Performance Computing. 2009;.

[9] Kolb C, Pharr M. Option pricing on the GPU in GPU gems 2. Addison-Wesley; 2005.

[10] Swarztrauber PN, Sweet RA. Vector and parallel methods for the direct solution of Poisson's equation. Journal of Computational and Applied Mathematics. 1989;27:241–263.

[11] Boyle PP. Option Valuation Using a Three-Jump Process. International Options Journal. 1986; 3:7–12.

[12] Rubinstein RY, Kroese DP. Simulation and the Monte Carlo Method. John Wiley and Sons; 2008.

[13] Gabriel E, Fagg GE, Bosilca G, et al. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In: Proceedings, 11th European PVM/MPI Users Group Meeting; Sept; Budapest, Hungary; 2004.

[14] R Core Team. R: A language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing; 2012. ISBN 3-900051-07-0; Available from: `http://www.r-project.org`.

[15] Yu H. Rmpi: Parallel statistical computing in R. R News. 2002;2(2):10–14. Available from: `http://cran.r-project.org/doc/Rnews/Rnews_2002-2.pdf`.

[16] Ostrouchov G, Chen WC, Schmidt D, et al. Programming with big data in R ; 2012. Available from: http://r-pbd.org/.

[17] Eddelbuettel D, François R. Rcpp: Seamless R and C++ integration. Journal of Statistical Software. 2011;40(8):1–18. Available from: http://www.jstatsoft.org/v40/i08/.

[18] Bezanson J, Edelman A, Karpinski S, et al. Julia: A fresh approach to numerical computing ; 2014. arXiv:1411.1607.

[19] Lattner C, Adve V. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In: Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04); Mar; Palo Alto, California; 2004.

[20] Noack A. Julia - MPI package [https://github.com/JuliaParallel/MPI.jl]; 2016.