Parallelizing Computation of Expected Values in a Binomial Tree with Bernoulli Paths

Sai K. Popuri*[1], Andrew M. Raim†[2], Nagaraj K. Neerchal[1], and Matthias K. Gobbert[1]

[1]Department of Mathematics and Statistics, University of Maryland, Baltimore County
[2]Center for Statistical Research & Methodology, U.S. Census Bureau

Abstract

Recombinant binomial trees are structures with two child nodes branching out from every node, including the root node, in a way that the nodes combine to form a symmetric structure. They arise in several model computations. For example, in option pricing in finance, valuation of a European option can be carried out by evaluating the expected value of the asset payoffs with respect to the probabilities of traversing the branches of the tree, when a closed form solution is not appropriate. When the size of the tree increases, the cost to compute the expected value grows exponentially, rendering a serial computation of one branch at a time very time-consuming and not practical. We propose a parallelization method that transforms the calculation of the expected value into a so-called 'embarrassingly parallel' problem by mapping the branches of the binomial tree to the processes on a parallel computing cluster. A Monte Carlo estimation method that benefits from the partitioning setup in our parallelization method is presented and the resulting variance reduction is discussed. The proposed parallelization method is implemented to price a European option in the statistical environment `R` and the programming language `Julia`. The numerical results indicate that while both the `R` and `Julia` implementations are scalable, the execution times of the `Julia` implementation are significantly better than the corresponding `R` implementation. A simulation study is carried out for an Asian and a Fixed Look-back option to verify the convergence and the variance reduction behavior in the proposed Monte Carlo method in relation to the basic Monte Carlo estimator.

Keywords: Binomial tree, Bernoulli paths, Parallel computing, Monte Carlo estimation, Path-dependent options.

1 Introduction

An N-step recombinant binomial tree is a binary tree where each non-leaf node has two children, which we will call "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 path (up, down) is assumed to be equivalent to the path (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

*For Correspondence: saiku1@umbc.edu

†Disclaimer: This article is released to inform interested parties of ongoing research and to encourage discussion of work in progress. Any views expressed are those of the authors and not necessarily those of the U.S. Census Bureau.

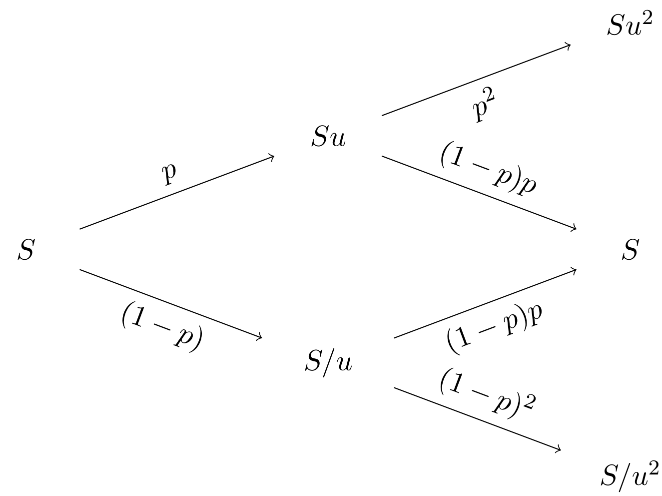


Figure 1.1: A two-step recombinant binomial tree.

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.

A primary example of recombinant binomial trees is the binomial options pricing model proposed in Cox et al. (1979). This model accounts for uncertainty of a future stock price based on 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 paths (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 expected option payoff discounted to the present value or is calculated by traversing the tree backwards and revising the option value at each step. See Hull (2003) and Seydel (2003) for more details on options and their valuation. In this paper we consider the case of European options, where a backward traversal of the tree is not required. 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 (Little, 1993), 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. For example, in a patient survey, sometimes a caregiver to the patient might respond to survey questions on behalf of the patient (Hosseini et al., 2016). Let $\boldsymbol{z}_i = (z_{i1}, \ldots, z_{iT})$ be the vector of Bernoulli random variables, where z_{it} is 0 if the respondent is the caregiver and 1 if the patient is the respondent. The log-likelihood for such a model is given by the mixture form

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

where f is the conditional density of $\boldsymbol{y}_i$ given $\boldsymbol{z}_i$, g is the distribution of the pattern of Bernoulli random variables, and $\boldsymbol{\theta}$ is the parameter. Here, the pattern $\boldsymbol{z}_i$ could be viewed as a Bernoulli path in a recombinant binomial tree.

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 propose a method to parallelize the calculation in a multiprocessor computing environment.

Parallelization of options pricing was considered by Popuri et al. (2013), who proposed a "master-worker" paradigm. Here, a master process partitions the set $\mathbb{B}^N$ and allocates the subsets to worker processes. The final answer is calculated by collecting the worker-level expected values. This paper instead uses a Single Program Multiple Data (SPMD) approach (Pacheco, 1997), where each of the M processes determines its assigned subset of $\mathbb{B}^N$ without coordination from a central master. Hence, the calculation can be transformed into an "embarrassingly parallel" problem (Foster, 1995), in which processes need not communicate except at the end of the computation. This avoids most of the overhead seen in Popuri et al. (2013) and allows 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 a European option using the 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 European put options in `R` and `Julia`. Concluding remarks are given in Section 6.

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

A European 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. We will assume European options in this paper unless otherwise specified.[1] A call option

[1] TBD: Andrew asks. . . Is it safe to just assume "European" for this section? In other sections, we seem to depart from that.

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. Denote by T the time to the future date of maturity or expiration, after which, the option is worthless. The value of a call option is the amount a buyer is willing to pay when the option is bought. Likewise, the value of a put option is the amount a selling is looking to receive when the option is sold. The value 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 $V(S_T, T)$,[2] 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 $V(S_T, T)$ 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).[3]

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

[2] TBD: Andrew asks…the notation suddenly changes here so that $V(\cdot, \cdot)$ takes two arguments. Is this consistent?

[3] TBD: Andrew asks…Is it appropriate to use "payoff" in place of "value"? Is this okay for both calls and puts?

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

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

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

The standard deviation σ is also known as the volatility of the stock. For more details on deriving u, d, and p see Hull (2003) or Seydel (2003).

Starting with the current stock price in the market S_0, a grid of possible future stock prices S_{ij} is built using u, d 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.[4] 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 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

$$V(S_0) = e^{-qT} \sum_{i=0}^{N} p(i) V_{Ni} = e^{-qT} \sum_{i=0}^{N} \binom{N}{i} p^i (1-p)^{N-i} V_{Ni}, \tag{2.3}$$

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

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

[4] TBD: Andrew asks…What does “each j” mean? Was j index introduced yet?

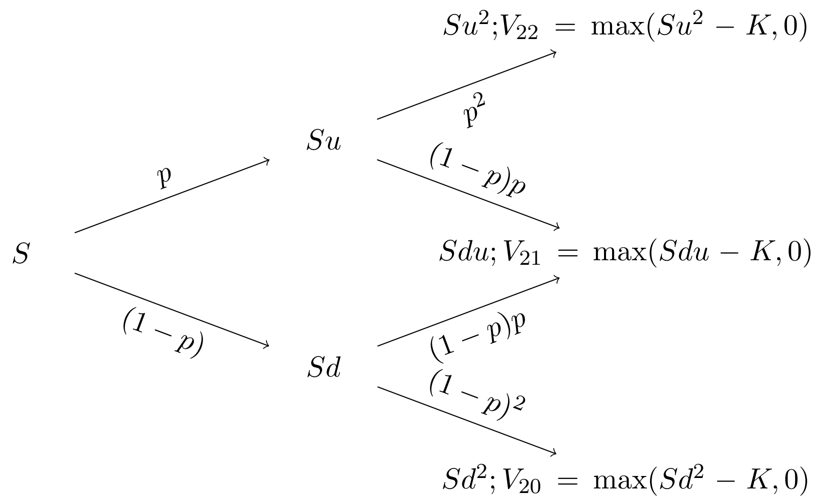


Figure 2.1: A two-step recombinant binomial tree with option payoffs.

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

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

Substituting (2.4) in (2.3), we obtain

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

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

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

where $p(\boldsymbol{x}) = \prod_{i=1}^{N} p_i^{I(x_i=1)}(1-p_i)^{I(x_i=0)}$. 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).

3 Parallel Bernoulli Path Algorithm

Even though the computation of the expected value in (2.6) is straight-forward, it becomes prohibitively expensive even for modest values of N. Since there are 2^N Bernoulli paths in a recombinant binary tree of size N, the computation of the expected value is of order 2^N. For example,

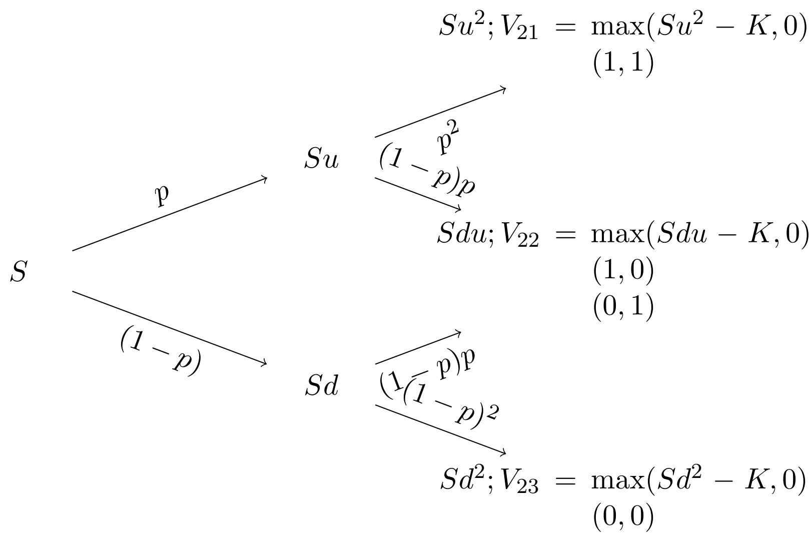


Figure 2.2: Two Step binomial Tree with Bernoulli Paths

for $N = 24$, this amounts to 16,777,216 paths! The valuation burden can be greatly shared if the expected value is transformed into a so-called 'embarrassingly parallel' problem. Although there are parallel methods proposed for evaluating the backward induction binary tree option pricing models (Ganesan et al., 2009; Kolb and Pharr, 2005), to the best of our knowledge, there are no methods to parallelize an expected value involving Bernoulli paths as formulated in (2.6). We propose a scheme that maps the Bernoulli paths to available processes. In our method, the rank of each process in the requested pool of processes uniquely determines a set of Bernoulli paths among the 2^N paths and each process computes the corresponding expected value. In the parallel computing parlance, the final expected value is computed by 'reducing' process level expected values to calculate the value in equation (2.6). For example, the Message Passing Interface (MPI) framework offers a "reduce" operation that collects the computed values from all processes and calculates the final value by performing an arithmetic operation (e.g., summation) on the collected values in an efficient way (Pacheco, 1997).

Our algorithm is based on the SPMD paradigm, where a single program is executed on all the processes in parallel. Individual processes collaborate with each other to execute parts of the program. This is in contrast to the 'master-worker' paradigm (used in Popuri et al. (2013)), where the master process builds the tree, calculates the payoffs, allocates the terminal nodes to the worker processes, and collects the calculated values from each worker process to put together the final answer. Even though the processes do not communicate with each other during the calculation, there is substantial initial communication between the master and the worker processes. In our algorithm, there is no initial allocation of payoffs to processes. Instead, each process determines a unique share of the 2^N paths to work on.

Let m be the rank of the process and M be the total number of processes employed. In the MPI framework, m ranges from 0 to $M - 1$. We assume that $M \leq N$ and that M is a power of 2. We further assume that the elements X_i of each Bernouli path $\boldsymbol{X}$ are independent Bernoulli random variables with parameter $p_i = \mathrm{P}(X_i = 1)$. In our parallelization method, each process works on 2^{N-r} paths, where $r = \log_2(M)$. Process m assembles the Bernoulli paths that it will work on by

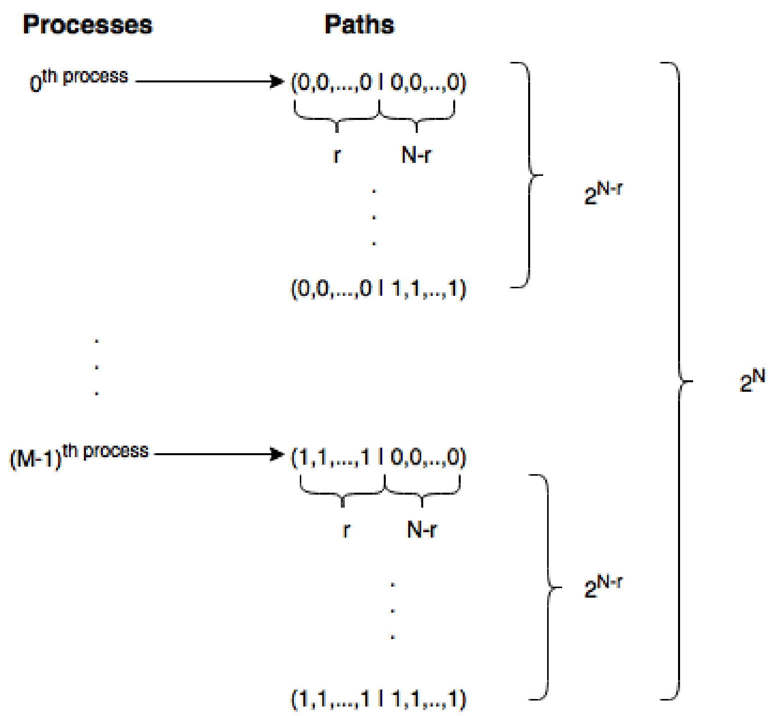


Figure 3.1: Process-Bernoulli path mapping

prepending the r-digit binary representation of the integer m to each of the $(N-r)$-digit binary representation of integers from 0 to $2^{N-r}-1$. Figure 3.1 shows a diagramatic representation of the mapping scheme. For each such Bernoulli path constructed $\boldsymbol{x}_{mi}$, where $i = 0, \ldots, 2^{N-r}-1$, the probability of traversing the path $p(\boldsymbol{x}_{mi})$ and the payoff value $V_N(\boldsymbol{x}_{mi})$ are calculated. Let $\mathbb{B}_m^N$ be the set of Bernoulli paths with cardinality 2^{N-r} that process m maps to itself. The expected value of the payoffs $V_N(\boldsymbol{x}_m)$ on process m is calculated as

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

Expected values in (3.1) are calculated on all the processes and are 'reduced' to the process 0 to calculate the final answer:

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

Note that in contrast to the 'master-worker' paradigm, the process 0 does not assume the onus of building the tree, and allocating, and communicating the terminal nodes to the other processes. In our algorithm, the process 0, in addition to computing its own expected value, merely collects the expected values from all other processes to calculate the final value. Note that the computation of the expected value in (3.2) is approximately of order 2^{N-r}, a reduction in the execution time by a factor of 2^r or M.

4 Monte Carlo Estimation and Variance Reduction

Since the number of Bernoulli paths in the set $\mathbb{B}^N$ grows exponentially with N, instead of calculating the expected value in (2.6) over all 2^N paths, one could estimate it using the Monte Carlo (MC) estimation method. Furthermore, because our parallelization method partitions $\mathbb{B}^N$, a separate MC estimation can be carried out on each process and the overall MC estimate can then be calculated from the process level estimates. We call this method the partitioned MC method. Besides, since the MC estimation is carried out in disjoint subregions of $\mathbb{B}^N$, the variance of the partitioned MC estimator is typically less than the variance of the basic MC estimator; see Rubinstein and Kroese (2008). In this section, we describe the partitioned MC estimation based on our parallelization method and verify the associated variance reduction.

Let θ be the expected value in (2.6):

$$\theta = \sum_{\boldsymbol{x} \in \mathbb{B}^N} V(\boldsymbol{x}) p(\boldsymbol{x}),$$

where the suffix N in $V(\boldsymbol{x})$ is dropped for notational convenience. The option value in (2.6) can then be written as

$$V = e^{-qT}\theta. \tag{4.1}$$

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

$$\hat{\theta} = \frac{1}{R}\sum_{i=1}^{R} V(\boldsymbol{x}_i), \tag{4.2}$$

whose variance is given by

$$\operatorname{Var}(\hat{\theta}) = \frac{1}{R}\operatorname{Var}[V(\boldsymbol{X})]. \tag{4.3}$$

Therefore, the MC estimator of the option value in (4.1) is given by

$$\hat{V} = e^{-qT}\hat{\theta}, \tag{4.4}$$

whose variance is $\frac{e^{-2qT}}{R}\operatorname{Var}[V(\boldsymbol{X})]$. This variance can be estimated by

$$\widehat{\operatorname{Var}}(\hat{V}) = \frac{e^{-2qT}}{R}\widehat{\operatorname{Var}}[V(\boldsymbol{X})]. \tag{4.5}$$

Notice that our parallelization method partitions the 'population' of 2^N Bernoulli paths into M subregions, with each containing 2^{N-r} Bernoulli paths whose first r elements constitute the binary representation of the corresponding rank of the process. Let $\mathcal{D}_m$ denote the event $[\boldsymbol{X} \in \mathbb{B}^N_m]$, $m = 0, \ldots, M-1$. Let the Bernoulli path $\boldsymbol{X}$ be partitioned as $(\boldsymbol{Z}, \boldsymbol{Y})$, where $\boldsymbol{Z} \in \mathbb{B}^r$ and $\boldsymbol{Y} \in \mathbb{B}^{N-r}$. We can now write θ as

$$\theta = \operatorname{E}[V(\boldsymbol{X})] = \sum_{m=0}^{M-1} \operatorname{E}[V(\boldsymbol{X}) \mid \mathcal{D}_m] \operatorname{P}(\mathcal{D}_m), \tag{4.6}$$

where

$$\mathrm{P}(\mathcal{D}_m) = \mathrm{P}(\boldsymbol{X} \in \mathbb{B}_m^N) = \sum_{\boldsymbol{y} \in \mathbb{B}^{N-r}} \mathrm{P}(\boldsymbol{Z} = \boldsymbol{z}_m, \boldsymbol{Y} = \boldsymbol{y}) = \mathrm{P}(\boldsymbol{Z} = \boldsymbol{z}_m), \tag{4.7}$$

and $\boldsymbol{z}_m$ is the binary representation of m, the rank of the m^{th} process and the last equality is possible because $\boldsymbol{Z}$ and $\boldsymbol{Y}$ are independent. Let $\theta_m = \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}_m]$ and let $(\boldsymbol{x}_{m1}, \ldots, \boldsymbol{x}_{mR_m})$ be an i.i.d. sample from $\mathbb{B}_m^N$, $m = 0, \ldots, M-1$ such that $\sum_{m=0}^{M-1} R_m = R$, the sample size in the basic MC estimator. Then, the MC estimator of θ_m is given by

$$\hat{\theta}_m = \frac{1}{R_m} \sum_{j=1}^{R_m} V(\boldsymbol{x}_{jm}),$$

which is an unbiased estimator of θ_m with variance $\frac{1}{R_m} \mathrm{Var}[V(\boldsymbol{X}) \mid \mathcal{D}_m]$. Substituting $\hat{\theta}_m$ for θ_m in (4.6), we get the partitioned MC estimator of θ as

$$\hat{\theta}_s = \sum_{m=0}^{M-1} \hat{\theta}_m \, \mathrm{P}(\mathcal{D}_m). \tag{4.8}$$

Therefore the partitioned MC estimator of the option value in (4.1) is given by

$$\hat{V}_s = e^{-qT} \hat{\theta}_s. \tag{4.9}$$

Following Rubinstein and Kroese (2008), we choose sample sizes R_m proportional to $\mathrm{P}(\mathcal{D}_m)$ as $R_m = R \, \mathrm{P}(\mathcal{D}_m)$, for each m. With this sample size, the variance of the partitioned MC estimator in (4.8) can be written as

$$\begin{aligned} \mathrm{Var}(\hat{\theta}_s) &= \sum_{m=0}^{M-1} \mathrm{Var}(\hat{\theta}_m)[\mathrm{P}(\mathcal{D}_m)]^2 \\ &= \frac{1}{R} \sum_{m=0}^{M-1} \mathrm{Var}[V(\boldsymbol{X}) \mid \mathcal{D}_m][\mathrm{P}(\mathcal{D}_m)]. \end{aligned} \tag{4.10}$$

The corresponding variance estimator of $\hat{V}_s$ is

$$\widehat{\mathrm{Var}}(\hat{V}_s) = \frac{e^{-2qT}}{R} \sum_{m=0}^{M-1} \widehat{\mathrm{Var}}[V(\boldsymbol{X}) \mid \mathcal{D}_m][\mathrm{P}(\mathcal{D}_m)]. \tag{4.11}$$

We now verify the variance reduction in the partitioned MC estimator. From the law of total variation, we have

$$\begin{aligned} \mathrm{Var}[V(\boldsymbol{X})] &= E_{\mathcal{D}} \, \mathrm{Var}[V(\boldsymbol{X}) \mid \mathcal{D}] + \mathrm{Var}_{\mathcal{D}} \, \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}] && (4.12) \\ &= \sum_{m=0}^{M-1} \mathrm{Var}[V(\boldsymbol{X}) \mid \mathcal{D}_m] \, \mathrm{P}(\mathcal{D}_m) + \mathrm{Var}_{\mathcal{D}} \, \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}] \\ &= R \, \mathrm{Var}(\hat{\theta}_s) + \mathrm{Var}_{\mathcal{D}} \, \mathrm{E}[V(\boldsymbol{X}) \mid \mathcal{D}]. && \text{(from (4.10))} \end{aligned}$$

Substituting the left hand side in (4.12) in terms of $\mathrm{Var}(\hat{\theta})$ from (4.3) and dividing both sides by R we get

$$\mathrm{Var}(\hat{\theta}) = \mathrm{Var}(\hat{\theta}_s) + \frac{\mathrm{Var}_{\mathcal{D}}\, E[V(\boldsymbol{X}) \mid \mathcal{D}]}{R}. \tag{4.13}$$

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 $r = 0$ or N, that is, when the number of processes $M = 1$ or 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 (4.13) is greater than 0 when $0 < r < N$ and therefore, the partitioned MC estimator in (4.8) typically yields strict reduction in the variance. We note that the variance reduction would be more pronounced when $V(\boldsymbol{X})$ are heterogeneous across the subregions $\mathcal{D}_m$ and homogeneous within each $\mathcal{D}_m$, $m = 0, \ldots, M-1$.

Remark 1. An interesting special case of the partitioned MC estimation method is to use a single sample of size R from $\mathbb{B}^{N-r}$ in a conditional expectation of $V(X)$. Consider θ in the form shown in (4.6). Let $\boldsymbol{y}_i$, $i = 1, \ldots, R$ are i.i.d. Bernoulli vectors from $\mathbb{B}^{N-r}$ and let $\boldsymbol{x}_{im} = (\boldsymbol{z}_m, \boldsymbol{y}_i)$, where $\boldsymbol{z}_m$ is the binary representation of m, the rank of the m^{th} process. Then θ in (4.6) can be estimated by

$$\begin{aligned}
\tilde{\theta} &= \sum_{m=0}^{M-1} \Big[\frac{1}{R}\sum_{i=1}^{R} V(\boldsymbol{x}_{im})\Big]\,\mathrm{P}(\mathcal{D}_m) \\
&= \frac{1}{R}\sum_{i=1}^{R}\sum_{m=0}^{M-1} V(\boldsymbol{x}_{im})\,\mathrm{P}(\mathcal{D}_m) \\
&= \frac{1}{R}\sum_{i=1}^{R} \mathrm{E}_{\boldsymbol{X}|\boldsymbol{Y}}[V(\boldsymbol{Z}, \boldsymbol{y}_i)].
\end{aligned} \tag{4.14}$$

The MC estimator $\tilde{\theta}$ in (4.14) is unbiased and has the variance $\frac{1}{R}\,\mathrm{Var}_{\boldsymbol{Y}}\,\mathrm{E}_{\boldsymbol{X}|\boldsymbol{Y}}[V(\boldsymbol{X})]$. The corresponding MC estimator of the option value in (4.1) can then be written as

$$\tilde{V} = e^{-qT}\tilde{\theta} \tag{4.15}$$

and the corresponding variance estimator is

$$\widehat{\mathrm{Var}}(\tilde{V}) = e^{-2qT}\widehat{\mathrm{Var}}(\tilde{\theta}). \tag{4.16}$$

We refer to this estimator as the partial MC estimator, since the sample is from $\mathbb{B}^{N-r}$ instead of $\mathbb{B}^N$. Now, the variance of $V(\boldsymbol{X})$ can be written as

$$\begin{aligned}
\mathrm{Var}[V(\boldsymbol{X})] &= \mathrm{Var}_{\boldsymbol{Y}}\,\mathrm{E}[V(\boldsymbol{X}) \mid \boldsymbol{Y}] + \mathrm{E}_{\boldsymbol{Y}}\,\mathrm{Var}[V(\boldsymbol{X}) \mid \boldsymbol{Y}] \\
&= R\,\mathrm{Var}(\tilde{\theta}) + \mathrm{E}_{\boldsymbol{Y}}\,\mathrm{Var}[V(\boldsymbol{X}) \mid \boldsymbol{Y}].
\end{aligned} \tag{4.17}$$

Substituting the left hand side in (4.17) in terms of $\mathrm{Var}(\hat{\theta})$ from (4.3) and dividing both sides by R we get

$$\mathrm{Var}(\hat{\theta}) = \mathrm{Var}(\tilde{\theta}) + \frac{\mathrm{E}_{\boldsymbol{Y}}\,\mathrm{Var}[V(\boldsymbol{X}) \mid \boldsymbol{Y}]}{R}. \tag{4.18}$$

Again, the second term in the right hand side of the (4.18) is zero if and only if $V(\boldsymbol{X}) \mid \boldsymbol{Y}$ does not depend on the first r steps of the Bernoulli paths, when $0 < r < N$. Since we assume that

$V(\boldsymbol{X})$ depends on the entire path $\boldsymbol{X}$, $\operatorname{Var}(\tilde{\theta})$ is strictly less than $\operatorname{Var}(\hat{\theta})$. Note that this method is an implementation of the partitioned MC method with the same sample from $\mathbb{B}^{N-r}$ of size R used on all the processes. As a result, the variance of the partitioned MC estimator is typically less than the variance of $\tilde{\theta}$ when $R_m = R$, $m = 0, \dots, M-1$ and the payoffs on paths with the same last $N-r$ elements are positively correlated, as the following result shows.

Theorem 4.1. *If the sample size $R_m = R$, $m = 0, \dots, M-1$, $0 < r < N$, and $V(\boldsymbol{Z}_k, \boldsymbol{Y})$ and $V(\boldsymbol{Z}_l, \boldsymbol{Y})$ are positively correlated for all $k, l = 0, \dots, M-1$, $k \neq l$, then $\operatorname{Var}(\hat{\theta}_s) \leq \operatorname{Var}(\tilde{\theta})$.*

Proof. Since $R_m = R$ for $m = 0, \dots, M-1$, we have $\operatorname{Var}(\hat{\theta}_s) = \frac{1}{R}\sum_{m=0}^{M-1} \operatorname{Var}[V(\boldsymbol{X}) \mid \mathcal{D}_m][\mathrm{P}(\mathcal{D}_m)]^2$. From (4.14), we have

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

□

5 Application to European 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 the distributed cluster maya at the UMBC High Performance Computing Facility (HPCF). The cluster maya has 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 (Gabriel et al., 2004) 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 (R Core Team, 2012). 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` (Yu, 2002) and `pbdMPI` (Ostrouchov et al., 2012) 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` (Eddelbuettel and François, 2011) 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 (Bezanson et al., 2014). It is a compiled language that uses the Low Level Virtual Machine Just-in-Time technology (Lattner and Adve,

2004) 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 (Noack, 2016) 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 proposed 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 summmed 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: Julia implementation.

We take a European put option as an example to illustrate our results. 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 $N = 20, 24, 28$, 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 program for $N = 32$ on a single process ($M = 1$) resulted in an overflow in our 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. Figure 5.1 shows the speedup and efficiency of our program in `R` and Figure 5.2 shows the corresponding plots from our program in `Julia`. 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 (1997) for more details.

We implemented the Monte Carlo estimation methods described in Section 4 for two types of path-dependent options (Hull, 2003): Asian and Look-back options. 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 $\{S_t : t = 1, \ldots, T\}$.[5] 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 Look-back 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 Look-back 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 (4.4) for the Asian and Fixed Look-back put options using the binomial tree model with size N to study the convergence of the estimates to the exact expected value in (2.6). We further implemented the partitioned MC and its special case given in (4.9) and (4.15) respectively to study the relative variance reduction behaviors. Table 5.3 shows basic MC estimates with corresponding variance estimates in (4.5) of an Asian put option and a Fixed Look-back put option with parameters $K = 100$, $S = 20$, $q = 6\%$, $\sigma = 3.0$, and $T = 1$ implemented using the binomial model with tree size $N = 32$. Option values calculated as the exact expected values in (2.6) are 82.115 for the Asian put options and 93.196 for the Fixed Look-back 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

[5] TBD: Andrew asks. . . Should the last index there be N?

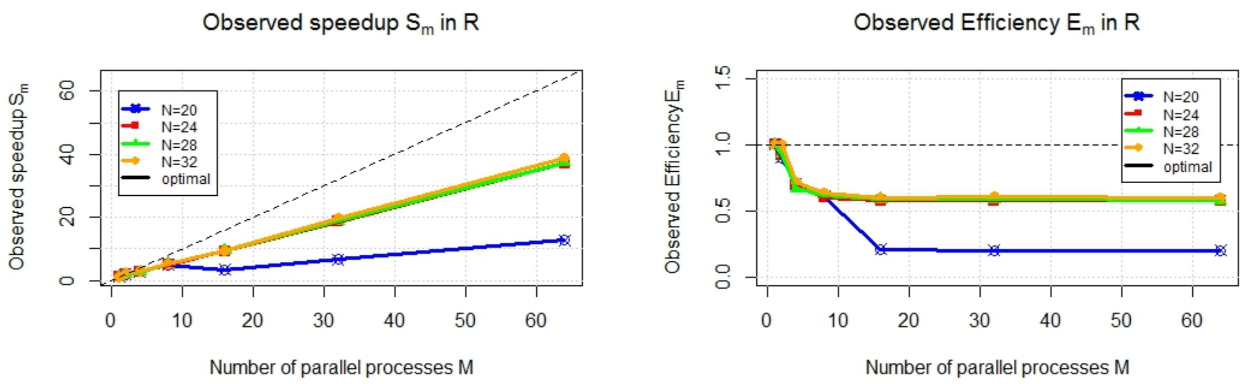
Table 5.1: Runtime for different number of time steps for R implementation. For $M = 1$, $N = 32$, since our program failed to run because of integer overflow, runtime is shown as N/A

(a) Wall clock time in HH:MM:SS							
N	$M = 1$	2	4	8	16	32	64
20	00:00:68	00:00:37	00:00:24	00:00:14	00:00:07	00:00:04	00:00:02
24	00:19:41	00:10:39	00:07:00	00:04:02	00:02:04	00:01:05	00:00:32
28	05:45:57	03:04:34	02:08:18	01:09:07	00:36:25	00:18:17	00:09:15
32	N/A	52:16:13	36:15:16	20:26:59	11:04:18	05:21:38	02:41:57
(b) Observed speedup S_M							
N	$M = 1$	2	4	8	16	32	64
20	1.00	1.81	2.79	4.92	3.38	6.61	12.80
24	1.00	1.85	2.81	4.88	9.30	18.58	37.22
28	1.00	1.87	2.69	5.00	9.49	18.90	37.38
32	1.00	2.00	2.88	5.11	9.44	19.50	38.72
(c) Observed efficiency E_M							
N	$M = 1$	2	4	8	16	32	64
20	1.00	0.90	0.70	0.62	0.21	0.21	0.20
24	1.00	0.92	0.70	0.61	0.58	0.58	0.58
28	1.00	0.94	0.67	0.62	0.60	0.60	0.58
32	1.00	1.00	0.72	0.64	0.59	0.61	0.60

shows the partitioned MC estimator for both the Asian and Fixed Look-back put options and the corresponding variances estimates using (4.11), for 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 partial MC estimates for the Asian put option with $N = 32$, $R = R_m = 1024$, and $m = 0, \ldots, M-1$. Partial MC estimates and corresponding variance estimates are calculated as in (4.15) and (4.16), respectively. 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 partial method does, as expected from the 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.

6 Concluding Remarks

We have proposed a novel 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 proposed a Monte Carlo estimation method which takes advantage of this partitioning. The proposed methods were implemented both in `R` and `Julia`, and were applied to value path-dependent European options. Numerical results verify the convergence of the proposed Monte Carlo method and variance reduction with respect to basic Monte Carlo estimation. Performance results indicate that the `Julia` implementation was



(a) (b)

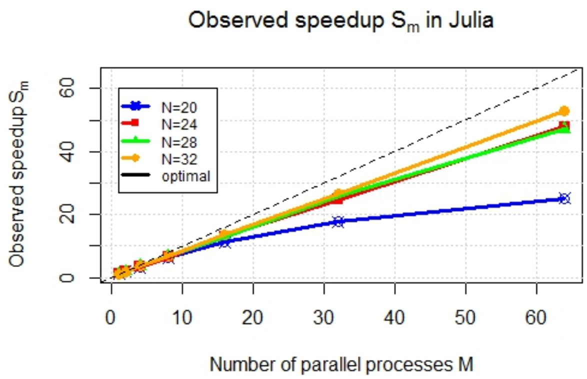
Figure 5.1: (a) Speedup and (b) Efficiency in R.

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

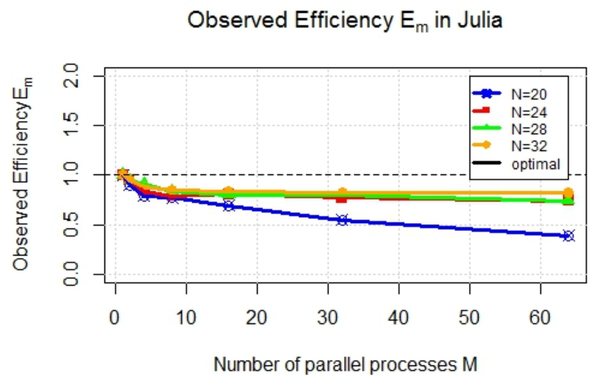
(a) Wall clock time in HH:MM:SS							
N	$M = 1$	2	4	8	16	32	64
20	00:00:07	00:00:05	00:00:02	00:00:01	00:00:01	00:00:01	00:00:01
24	00:02:23	00:01:15	00:00:42	00:00:22	00:00:11	00:00:06	00:00:03
28	00:40:08	00:21:05	00:11:02	00:05:56	00:03:07	00:01:34	00:00:51
32	11:59:54	06:23:01	03:24:58	01:46:06	00:53:41	00:27:18	00:13:38
(b) Observed speedup S_M							
N	$M = 1$	2	4	8	16	32	64
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
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

Table 5.3: Monte Carlo estimates and corresponding variance estimates for Asian and Fixed Look-back put options with $N = 32$. Exact value of the Asian option is 82.115 and the Look-back 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
	$\widehat{\mathrm{Var}}(\hat{V})$	0.735	0.362	0.179	0.095	0.050	0.022	0.011	0.006
Look-back	$\hat{V}$	93.156	93.237	93.312	93.324	93.262	93.236	93.234	93.222
	$\widehat{\mathrm{Var}}(\hat{V})$	0.022	0.009	0.005	0.002	0.001	<0.001	<0.001	<0.001



(a)



(b)

Figure 5.2: (a) Speedup and (b) Efficiency in Julia.

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

Option	Estimate	$M = 1$	2	4	8	16	32	64
		$R_m = 2^{10}$	2^9	2^8	2^7	2^6	2^5	2^4
Asian	$\hat{V}_s$	82.077	82.217	82.101	82.232	81.936	82.296	82.165
	$\widehat{\mathrm{Var}}(\hat{V}_s)$	0.367	0.332	0.315	0.272	0.263	0.212	0.194
Look-back	$\hat{V}_s$	93.196	93.201	93.171	93.197	93.187	93.216	93.205
	$\widehat{\mathrm{Var}}(\hat{V}_s)$	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 partial 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
	$\widehat{\mathrm{Var}}(\hat{V}_s)$	0.367	0.220	0.062	0.027	0.005	0.005	0.002
Partial MC	$\tilde{V}$	82.113	82.112	82.120	82.093	82.102	82.124	82.108
	$\widehat{\mathrm{Var}}(\tilde{V})$	0.373	0.305	0.267	0.203	0.170	0.142	0.123

significantly faster and more efficient than the `R` implementation, likely because of the superior handling of looping and the compilation technology in `Julia`.

Acknowledgments

The first author acknowledges financial support from the UMBC High Performance Computing Facility (HPCF) at the University of Maryland, Baltimore County (UMBC). The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant no. CNS–0821258 and CNS–1228778) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See hpcf.umbc.edu for more information on HPCF and the projects using its resources.

References

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A Fresh Approach to Numerical Computing. *ArXiv e-prints*, November 2014.

John C. Cox, Stephen A. Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of Financial Economics*, 7(3):229–263, 1979. ISSN 0304-405X. doi: http://dx.doi.org/10.1016/0304-405X(79)90015-1. URL http://www.sciencedirect.com/science/article/pii/0304405X79900151.

Dirk Eddelbuettel and Romain François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL http://www.jstatsoft.org/v40/i08/.

Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley, 1995.

Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users Group Meeting*, Budapest, Hungary, Sept 2004.

Narayan Ganesan, Roger D. Chamberlain, and Jeremy Buhler. Acceleration of binomial options pricing via parallelizing along time-axis on a GPU. *Proc. of Symp. on Application Accelerators in High Performance Computing*, 2009.

Mina Hosseini, Nagaraj K. Neerchal, and Ann L. Gruber-Baldini. Statistical modeling of subject and proxy observations using weighted GEE. In *JSM Proceedings, Section on Statistics in Epidemiology. Alexandria, VA: American Statistical Association*, pages 1101–1111, 2016.

John C. Hull. *Options, Futures, And Other Derivatives.* Prentice Hall, 2003.

C. Kolb and M. Pharr. *Option pricing on the GPU in GPU Gems 2.* Addison-Wesley, 2005.

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

Roderick J. A. Little. Pattern-mixture models for multivariate incomplete data. *Journal of the American Statistical Association*, 88(421):125–134, 1993.

Andreas Noack. Julia - MPI package. https://github.com/JuliaParallel/MPI.jl, 2016.

G. Ostrouchov, W.-C. Chen, D. Schmidt, and P. Patel. Programming with big data in r, 2012. URL http://r-pbd.org/.

Peter S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann, 1997.

Sai K. Popuri, Andrew M. Raim, Nagaraj K Neerchal, and Matthias K. Gobbert. An implementation of binomial method of option pricing using parallel computing. Technical Report Technical Report HPCF–2013–1, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2013. URL http://userpages.umbc.edu/~gobbert/papers/PopuriEtAl_Binomial.pdf.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.r-project.org. ISBN 3-900051-07-0.

Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method*. John Wiley and Sons, 2008.

Rüdiger Seydel. *Tools for Computational Finance*. Springer, 2003.

Hao Yu. Rmpi: Parallel statistical computing in R. *R News*, 2(2):10–14, 2002. URL http://cran.r-project.org/doc/Rnews/Rnews_2002-2.pdf.