Fast American Option Pricing using Nonlinear Stencils

Zafar Ahmad
Stony Brook University
Stony Brook, USA
zafahmad@cs.stonybrook.edu

Reilly Browne
Stony Brook University
Stony Brook, USA
rjbrowne@cs.stonybrook.edu

Rezaul Chowdhury
Stony Brook University
Stony Brook, USA, USA
rezaul@cs.stonybrook.edu

Rathish Das
University of Houston
Houston, USA
rathish@uh.edu

Yushen Huang
Stony Brook University
Stony Brook, USA
yushuang@cs.stonybrook.edu

Yimin Zhu
Stony Brook University
Stony Brook, USA
yimzhu@cs.stonybrook.edu

Abstract

We study the binomial, trinomial, and Black-Scholes-Merton models of option pricing. We present fast parallel discrete-time finite-difference algorithms for American call option pricing under the binomial and trinomial models and American put option pricing under the Black-Scholes-Merton model. For T-step finite differences, each algorithm runs in $O\left(\left(T\log^2 T\right)/p + T\right)$ time under a greedy scheduler on p processing cores, which is a significant improvement over the $\Theta\left(T^2/p\right) + \Omega\left(T\log T\right)$ time taken by the corresponding state-of-the-art parallel algorithm. Even when run on a single core, the $O\left(T\log^2 T\right)$ time taken by our algorithms is asymptotically much smaller than the $\Theta\left(T^2\right)$ running time of the fastest known serial algorithms. Implementations of our algorithms significantly outperform the fastest implementations of existing algorithms in practice, e.g., when run for $T \approx 1000$ steps on a 48-core machine, our algorithm for the binomial model runs at least 15× faster than the fastest existing parallel program for the same model with the speedup factor gradually reaching beyond 500× for $T \approx 0.5 \times 10^6$. It saves more than 80% energy when $T \approx 4000$, and more than 99% energy for $T > 60,000$.

Our algorithms can be viewed as solving a class of nonlinear 1D stencil (i.e., finite-difference) computation problems efficiently using the Fast Fourier Transform (FFT). To our knowledge, ours are the first algorithms to handle such stencils in $o\left(T^2\right)$ time. These contributions are of independent interest as stencil computations have a wide range of applications beyond quantitative finance.

Full Version: `https://arxiv.org/abs/2303.02317`

ACM ISBN 979-8-4007-0435-2/24/03.
https://doi.org/10.1145/3627535.3638506

CCS Concepts: • **Mathematics of computing → Solvers**; • **Theory of computation → Shared memory algorithms**.

Keywords: American Option Pricing, Binomial Option Pricing Model, Trinomial Option Pricing, Black-Scholes-Merton Option Pricing Model, Nonlinear Stencil, Fast Fourier Transform, Finite-Difference Method

1 Introduction

Option pricing or computing the value of a contract giving one the right to buy/sell an asset under some given constraints is one of the most important computational problems in quantitative finance [6]. Rapid changes in financial markets often lead to rapid changes in asset prices which makes the ability to quickly estimate option prices essential in avoiding potential financial losses [51].

An ***option*** is a two-party financial contract that gives one party (called the ***holder***) the right (but not an obligation) to buy/sell (i.e., exercise) an asset from/to the other party (called the ***writer***) at a fixed price (called the ***strike/exercise price***) on or before an expiration date (called the ***exercise/maturity date***). A ***call option*** gives the right to buy whereas a ***put option*** gives the right to sell. Also, based on the expiration date and the settlement rule, there are two major styles of options: ***European*** and ***American***. A European option can only be exercised at the expiration date while an American option can be exercised at any time before that.

The ***option pricing*** problem asks for assigning a ***value*** or ***price*** to an options contract based on the calculated probability that the contract will be exercised at expiration. The theoretical value of an option [16, 18, 42, 62, 75, 101] is determined by its ***stock price*** S (i.e., its current market price), ***strike price*** K, ***risk-free rate of return*** R (i.e., the theoretical rate of return assuming zero risk), ***dividend yield*** Y (i.e., a ratio that shows how much dividend/year is paid relative to S), ***volatility*** V (i.e., how much the trading price varies over time), and ***time to expiration*** E (e.g., in days).

The earliest option pricing model [11] was based on the assumption of the geometric Brownian motion for asset pricing and the no-arbitrage idea. Many improved models were developed later [12, 13, 17, 18, 34–36, 38, 39, 39, 44, 46, 54, 60, 70, 74–76, 94, 95, 103].

Table 1. Notations

Symbol	Meaning	Symbol	Meaning
S	stock price	K	strike price
R	risk-free rate of return	V	volatility
Y	dividend yield	E	time to expire (in days)
T	number of time steps		

```
BOPM-American-Call( $S, K, R, V, Y, E, T$ )
  1. $\Delta t \leftarrow E/T$, $u \leftarrow e^{V\sqrt{\Delta t}}$, $d \leftarrow 1/u$, $p \leftarrow (e^{(R-Y)\Delta t} - d)/(u - d)$
     $m \leftarrow e^{-R\Delta t}$, $s_0 \leftarrow mp$, $s_1 \leftarrow m(1-p)$
  2. for $j \leftarrow 0$ to $T$ do $G_{T,j} \leftarrow \max\left(0, Su^{2j-T} - K\right)$
  3. for $i \leftarrow T-1$ downto $0$ do
         parallel for $j \leftarrow 0$ to $i$ do
             $G_{i,j} \leftarrow \max\left(s_0 G_{i+1,j} + s_1 G_{i+1,j+1},\ Su^{2j-i} - K\right)$
  4. return $G_{0,0}$
```

Figure 1. Standard looping code for pricing American Call options under the Binomial Option Pricing Model.

Analytical solutions to the option pricing problem are sometimes available, particularly for European options [52, 86, 99, 100]. But they are not available for American options except for a limited number of cases with significant constraints (e.g., American call options with zero or one dividend [100] and perpetual American put options [69, 100]). This difficulty in finding closed-form analytical solutions for most option pricing problems makes *computational* approaches the only path forward. The main computational approaches to solving the option pricing problem include the binomial tree method [58], the finite difference method [6, 50, 64, 97, 104, 110], and the Monte Carlo method [37, 66, 111].

The binomial tree method works by tracing the option's value at discrete time steps over the life of the option. For a given number of time steps T between the valuation and expiration dates of the option, a binomial tree of height T is created with the leaves storing the potential prices of the asset at the time of expiration. Then one works backward to compute for each $t \in [0, T-1]$ the value of the nodes at depth t of the tree (each giving a possible price at time step t) from the values of the nodes at depth $t+1$ using a simple formula. The value computed for the root node is the required option value. Straightforward iterative implementation of this method runs in $\Theta\left(T^2\right)$ time on a single processing core and $\Theta\left(T^2/p + T\log T\right)$ time on p cores (see Figure 1 and Table 2). It provides a discrete-time approximation of the continuous-time option pricing in the Black–Scholes model and is widely used by professional option traders.

The trinomial tree method extends the binomial method by accounting for the possibility that an asset value remains the same after a time step [21]. With only a constant factor increase in run-time it provides more precise predictions than the binomial model.

The finite-difference method approximates the continuous-time differential equations describing the evolution of an option price over time by a set of discrete-time difference

Table 2. Parallel Algorithms for American Option Pricing: The bounds hold for **call option pricing under the binomial and trinomial option pricing models**, and **put option pricing under the Black-Scholes-Merton model**. Here, T = number of time steps, p = number of processing cores, and M = cache size. Also, $\mathcal{T}_p$ = running time on p cores, and thus $\mathcal{T}_1$ (Work) and $\mathcal{T}_\infty$ (Span) represent run-times on one core and an unbounded number of cores, respectively. Under a greedy scheduler, $\mathcal{T}_p = \Theta\left(\mathcal{T}_1/p + \mathcal{T}_\infty\right)$, which is asymptotically optimal.

Algorithm	Work $\mathcal{T}_1(T)$	Span $\mathcal{T}_\infty(T)$	Parallel Running Time $\mathcal{T}_p(T)$
Nested Loop (standard, see Figure 1)	$\Theta\left(T^2\right)$	$\Theta\left(T\log T\right)$	$\Theta\left(\frac{T^2}{p} + T\log T\right)$
Tiled Loop (cache-aware) [23]		$\Theta\left(TM + \frac{T}{M}\log\frac{T}{M}\right)$	$\Theta\left(\frac{T^2}{p} + TM + \frac{T}{M}\log\frac{T}{M}\right)$
Recursive Tiling (cache-oblivious) [23, 40, 41, 107]		$\Theta\left(T^{\log_2 3}\right)$	$\Theta\left(\frac{T^2}{p} + T^{\log_2 3}\right)$
Our Algorithms	$\Theta\left(T\log^2 T\right)$	$\Theta\left(T\right)$	$\Theta\left(\frac{T\log^2 T}{p} + T\right)$

equations and then solves them iteratively under appropriate boundary conditions. The explicit finite difference method divides the lifetime of the option into T discrete time steps and then uses the potential values of the asset at time step T (the time of expiration) to compute the asset values at each time step $t \in [0, T-1]$ from the asset values at time step $t+1$ based on the difference equations (i.e., update equations or stencils). The final option value is found at time $t = 0$. Similar to the binomial tree method, the iterative implementation of this method runs in $\Theta\left(T^2\right)$ time on a single processing core and $\Theta\left(T^2/p + T\log T\right)$ time on p cores. Other finite difference methods used for option pricing include implicit finite difference and the Crank–Nicolson method [32].

The Monte Carlo method works by generating random backward paths the asset price may follow starting from the time of expiration and ending at the time of valuation. Each of these paths leads to a payoff value for the option and the average of these payoff values can be viewed as an expected value of the option. This method is used for pricing options with complicated features and/or multiple sources of uncertainty that other methods (analytical, tree-based, finite difference) cannot handle [37], but is usually not competitive when those methods apply as the convergence rate of Monte Carlo method is sublinear [55, 77]. They are hard to develop for some options, such as Black-Scholes Model for the American put option, but still many results exist on Monte Carlo methods [22, 53, 66, 111].

Our Contributions. We present three shared-memory parallel algorithms for American option pricing – call option under the binomial and trinomial option pricing models and put option under the Black-Scholes-Merton model. All three algorithms run in $\Theta\left(\left(T\log^2 T\right)/p + T\right)$ time on p processing cores which is a significant improvement over the $\Omega\left(T^2/p + T\log T\right)$ time taken by the state-of-the-art parallel algorithms, where T is the number of time steps. When run on a single processing core they run in $\Theta\left(T \log^2 T\right)$ time compared to the $\Theta\left(T^2\right)$ time taken by the fastest existing serial algorithms. We use the ***Fast Fourier Transform (FFT)*** to speed up the computation. Table 2 summarizes the results.

We use the *work-span* model [30] to analyze the performance of parallel programs. Let $\mathcal{T}_p$ be the running time on a p-processor machine under a greedy scheduler. Then $\mathcal{T}_1$ and $\mathcal{T}_\infty$ are called *work* and *span*, respectively. The *parallel running time* $\mathcal{T}_p = \Theta\left(\mathcal{T}_1/p + \mathcal{T}_\infty\right)$.

The following proposition, which follows easily from the complexities given in Table 2, notes that each of our parallel algorithms runs asymptotically faster than the corresponding fastest existing parallel algorithm for every value of p.

Proposition 1.1. *Let $\mathcal{T}_p^{(old)}(T)$ be the running time of any existing algorithm from Table 2 on p cores, and let $\mathcal{T}_p^{(new)}(T)$ denote the same for our algorithm. Then for every (positive) value of p under a greedy scheduler: $\mathcal{T}_p^{(new)}(T) = o\left(\mathcal{T}_p^{(old)}(T)\right)$.*

We have implemented our algorithms and compared their running times, energy consumption, and cache performance with those of the option pricing implementations available in the Par-bin-ops framework [23] developed recently in 2022. Implementations of our algorithms run orders of magnitude faster, consume significantly less energy, and usually incur far fewer L1 cache misses than those implementations.

How Our Algorithms Differ from Existing FFT-based Option Pricing Algorithms. FFTs have been used for European option pricing before. European option pricing is simpler than American option pricing, e.g., the European version of the American option pricing algorithm shown in Figure 1 can be obtained by replacing the assignment $G_{i,j} \leftarrow \max\left(s_0 G_{i+1,j} + s_1 G_{i+1,j+1},\ S u^{2j-i} - K\right)$ in Step 3 with the simpler assignment $G_{i,j} \leftarrow s_0 G_{i+1,j} + s_1 G_{i+1,j+1}$. The absence of the '**max**' operator in this assignment makes an efficient evaluation of the doubly-nested loop in Step 3 easier.

Black, Scholes, and Merton [18, 75] showed that the European option can be calculated using a Parabolic PDE with infinite domain constraint. By using the Fourier transform, one gets an integral form for European options. To obtain the numerical value from the integral form, one uses numerical integration [24, 47] which can be sped up using FFTs.

There are also approximation results [25, 68, 80, 129] based on repeated Richardson extrapolation [90, 91] and FFT for numerical integration in American options. However, even if the extrapolation is repeated only for a constant number of times for an option that expires in E days, the approximation takes $\Omega\left((E/\Delta t)\, N \log N\right)$ time when N grid points are used to discretize the price of the underlying asset and $E/\Delta t$ exercise points are placed with every pair of consecutive exercise points being Δt days apart. Observing that $T = E/\Delta t$ corresponds to the number of discrete time steps in the finite difference formulation of the problem, we can rewrite the complexity as $\Omega\left(TN \log N\right)$. Usually, $N \geq T$ is used in practice [80], which reduces the complexity to $\Omega\left(T^2 \log T\right)$.

A major weakness of the existing FFT-based numerical integration approach above is that a closed-form expression for the characteristic function of the log-price must be known for the technique to work. However, our approach does not need to know such a closed-form expression as we apply FFT to speed up stencil/finite-difference computations and not numerical integration. Thus, our approach will work on a larger set of option pricing problems. We are not restricted to infinite-domain problems either [28].

Implications for Nonlinear Stencil Computations. Our option pricing algorithms can be viewed as solving a class of ***nonlinear 1D stencil computation problems*** asking to evolve a grid of size $\Theta(T)$ for T time steps, in $\Theta\left(T \log^2 T\right)$ work (i.e., serial time) or $\Theta\left(\left(T \log^2 T\right)/p + T\right)$ parallel time on p processing cores. As stencil algorithms, they are of independent interest.

A ***stencil*** is a pattern (equation) used to update the values of cells in a spatial grid and evolve the grid over a given number of time steps. The process of evolving cell values in the spatial grid according to a stencil is called a ***stencil computation*** [40]. The finite-difference method performs a stencil computation with an update equation derived from the differential equations used as a stencil. Stencils are widely used in various fields, including mechanical engineering [83, 87, 89, 105, 126], meteorology [10, 56, 57, 78, 92, 93], stochastic and fractional differential equations [67, 123, 124], chemistry [9, 45, 65, 79, 102], electromagnetics [8, 59, 106, 113], finance [27], and physics [15, 33, 43, 48, 73, 82, 109, 115, 116], image processing [85, 114, 118].

Standard stencil algorithms perform $\Theta(NT)$ work to evolve a grid of size N for T time steps, they include looping algorithms, tiled looping algorithms [7, 14, 19, 20, 49, 71, 72, 119–122, 125], and recursive divide-and-conquer algorithms [40, 41, 61, 81, 96, 107, 108].

A stencil is called ***linear*** if it computes the value of a cell at time step t as a fixed linear combination of cell values at time steps before t, otherwise it is called ***nonlinear***. For ***1D linear stencils*** Ahmad et al. [2] provide FFT-based algorithms that take $O(T \log T)$ serial time for periodic grids and $O\left(T \log^2 T\right)$ serial time for aperiodic grids, assuming that the input grid is of size $\Theta(T)$.

The ***stencils we encounter in our current work are nonlinear*** because they do not use a linear combination of cell values from prior time steps for updating a target cell, provided that the resulting value is smaller than the value of a function computed solely based on the spatial coordinates of the target cell and other option pricing parameters (e.g., see Step 3 of Figure 1). Such a stencil divides the space-time grid into two disjoint regions – in one region only the linear combination applies, while in the other only the function value applies. However, the problem is that the boundary

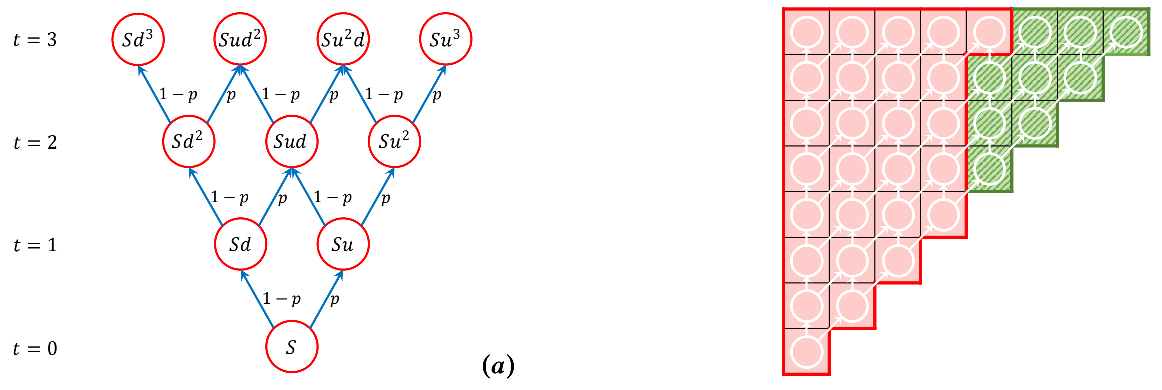


Figure 2. (a) A 3 time-step binomial tree. (b) A binomial tree of 7 time steps embedded in an 8×8 grid. An upward arrow has a price change factor of $1/u$ while a rightward arrow has a price change factor of u.

between these two regions is not known ahead of time and the location of the boundary may move as the time step t progresses. As a result, ***Ahmad et al.'s [2] results for linear stencils do not apply***. To the best of our knowledge, ours are the first algorithms for handling such stencils running in time subquadratic in T.

Organization of the Paper. The rest of the article is organized as follows. In Sections 2, 3, and 4 we describe our results for the binomial, trinomial, and Black-Scholes-Merton option pricing models, respectively. Experimental results comparing implementations of our algorithms with the state of the art are given in Section 5. In Section 6, we list a couple of example applications of our techniques in areas beyond quantitative finance. Finally, Section 7 concludes our paper.

2 American Call Option under the Binomial Option Pricing Model

2.1 Binomial Option Pricing Model (BOPM)

BOPM [31, 88, 98] is a simple discrete-time option pricing model without using advanced mathematical tools. It is a paradigm of practice.

BOPM encodes the various sequences of prices the asset might take as paths in a binomial tree. Each node in the tree represents a possible price at a certain time and the nodes at two successive layers in the tree represent prices at times that are apart by some fixed time step Δt. The prices increase or decrease by some factor after every Δt time. Figure 2(a) gives an example of a 3-time-step binomial price tree that is produced by moving from the valuation day to the expiration day. Denote the initial price by S. The price in the next time step (i.e., after Δt time) can go up to $f_u = S \cdot u$ or go down to $f_d = S \cdot d$, where the up factor $u = e^{V\sqrt{\Delta t}}$ and the down factor $d = 1/u$ are determined by Δt and volatility V.

Denote the node value as $X_{node} = S \times u^{N_u - N_d}$, where N_u and N_d are the numbers of ticks up and down, respectively. The final nodes of the tree represent the prices on the expiration date. Given the strike price of K, the price one can call or put before the contract expires, i.e., the ***exercise value*** of each node will be $\max(X_{node} - K, 0)$ for a call option and $\max(K - X_{node}, 0)$ for a put option.

The risk-neutral valuation of the binomial value is performed iteratively backward. Under the assumption of risk neutrality, the value of the option today is its expected future payoff discounted at the risk-free rate of R. Let us number the nodes in each layer of Figure 2(a) from top to bottom by successive integers starting from 1. Then the node values $X_{t,j}$ and $X_{t,j+1}$ of a layer representing some time t can be used to compute the ***binomial value*** of the j-th node in the layer representing time $t - \Delta t$ as follows: $e^{-R\Delta t}(p \cdot X_{t,j} + (1-p) \cdot X_{t,j+1})$, where, $p = (e^{R\Delta t} - d)/(u - d)$ [52]. Denote $m = e^{-R \cdot \Delta t}$, $s_0 = m(1-p)$, and $s_1 = mp$. Then the binomial value of that node is: $s_0 \cdot X_{t,j} + s_1 \cdot X_{t,j+1}$. For options on stocks paying a continuous dividend yield Y, $p = (e^{(R-Y)\Delta t} - d)/(u - d)$.

The value at each node will be equal to its binomial value for European options and the larger of its binomial value and exercise value for American options.

The binomial tree of T time steps can be embedded in a $(T+1) \times (T+1)$ grid. Figure 2(b) shows an example.

Definition 2.1. Let $G_{i,j}$ be the grid value in row $i \in [0, T]$ and column $j \in [0, T]$ of the $(T+1) \times (T+1)$ grid G. Let $G^{green}_{i,j} = S \cdot u^{2j-i} - K$, and let $G^{red}_{i,j} = s_0 G_{i+1,j} + s_1 G_{i+1,j+1}$ if $i \in [0, T)$, and 0 otherwise. Then

$$G_{i,j} = \begin{cases} G^{red}_{i,j}, & \text{if } G^{red}_{i,j} \geq G^{green}_{i,j} \\ G^{green}_{i,j}, & \text{otherwise.} \end{cases}$$

We say that cell (i, j) of G is ***red*** provided $G_{i,j} = G^{red}_{i,j}$, and ***green*** otherwise. We show in Section 2.2 that all red cells in G form a single contiguous region and all green cells form another. A single boundary divides the two regions. We analyze the properties of this ***red-green divider*** in Section 2.1 which we will exploit to design an efficient algorithm for American call options in Section 2.3.

2.2 Properties of the Red-Green Divider

As shown in the example in Figure 2(b), we assume that a binomial tree for T time steps is embedded in a $(T+1)\times(T+1)$ grid G with the root at the bottom-left corner $G[0,0]$ and the leaves in the top row $G[T,0..T]$. For $0 \le j \le i \le T$, the two children of the binomial tree node at $G[i,j]$ are stored at $G[i+1,j]$ and $G[i+1,j+1]$. The arrow from $G[i,j]$ to $G[i+1,j+1]$ represents a price change factor of u while the one from $G[i,j]$ to $G[i+1,j]$ represents a price change factor of $d = 1/u$. So, the entire tree occupies only the upper-left triangular part of the grid.

Lemma 2.2 shows that within the upper-left triangle of G, if a cell is green then the cell to its right is also green.

Lemma 2.2. $\left(G_{i,j+1} = G^{green}_{i,j+1}\right) \implies \left(G_{i,j} = G^{green}_{i,j}\right)$, *for* $i \in [0,T]$ *and* $j \in [0, i-1]$.

Proof. Observe that $\frac{s_0}{u} + s_1 u = mp\left(u - \frac{1}{u}\right) + \frac{m}{u} = e^{-Y\Delta t}$.

Let $\delta_{i,j} = G^{red}_{i,j} - G^{green}_{i,j}$, $\Delta_{i,j} = \delta_{i,j+1} - \delta_{i,j}$, $\tilde{\delta}_{i,j} = G_{i,j} - G^{green}_{i,j}$ and $\tilde{\Delta}_{i,j} = \tilde{\delta}_{i,j+1} - \tilde{\delta}_{i,j}$.

We will use mathematical induction to show that $\Delta_{i,j} \le 0$ for all $0 \le i \le T$ and $j < i$.

As $\delta_{T,j} = K - Su^{2j-T}$ and thus $\Delta_{T,j} = Su^{2j-T}\left(1-u^2\right) \le 0$, the claim holds for $i = T$.

Now suppose that the claim holds for some given $i+1 \le T$, i.e., $\Delta_{i+1,j} \le 0$ for $0 \le j < i+1 \le T$. Since $\Delta_{i+1,j} \le 0$, there exists a j_{i+1} such that $G_{i+1,j} = G^{green}_{i+1,j}$ when $j > j_{i+1}$ and $G_{i+1,j} = G^{red}_{i+1,j}$ when $j \le j_i$, where j_{i+1} is the largest j such that $\delta_{i+1,j} > 0$. Then,

- for $j > j_{i+1}$, $\tilde{\Delta}_{i+1,j} = \tilde{\delta}_{i+1,j+1} - \tilde{\delta}_{i+1,j} = 0 - 0 = 0$;
- for $j = j_{i+1}$, $\tilde{\Delta}_{i+1,j} = \tilde{\delta}_{i+1,j+1} - \tilde{\delta}_{i+1,j} = -\tilde{\delta}_{i+1,j} \le 0$; and
- for $j < j_{i+1}$, $\tilde{\Delta}_{i+1,j} = \Delta_{i+1,j} \le 0$.

Thus, $\tilde{\Delta}_{i+1,j} \le 0$ for all $j \in [0, i+1)$.
Hence, $\Delta_{i,j} = \delta_{i,j+1} - \delta_{i,j}$

$$= \sum_{k\in\{0,1\}} s_k \left(\left(G_{i+1,j+k+1} - G^{green}_{i+1,j+k+1}\right) - \left(G_{i+1,j+k} - G^{green}_{i+1,j+k}\right)\right)$$

$$+ \sum_{k\in\{0,1\}} s_k \left(G^{green}_{i+1,j+k+1} - G^{green}_{i+1,j+k}\right) + G^{green}_{i,j} - G^{green}_{i,j+1}$$

$$= s_0\tilde{\Delta}_{i+1,j} + s_1\tilde{\Delta}_{i+1,j+1} + Su^{2j-i}\left(e^{-Y\Delta t} - 1\right)\left(u^2 - 1\right) \le 0$$

Therefore, $\Delta_{i,j} \le 0$ for all $0 \le i < T$ and $j < i$.

Because $\Delta_{i,j} \le 0$, there exists a j_i such that when $j > j_i$, $G_{i,j} = G^{green}_{i,j}$ and when $j \le j_i$, $G_{i,j} = G^{red}_{i,j}$, where j_i is the largest j such that $\delta_{i,j} > 0$. Now if $G_{i,j} = G^{green}_{i,j}$, we have $j > j_i$, and thus $j+1 > j_i$, which implies that $G_{i,j+1} = G^{green}_{i,j+1}$. □

Lemma 2.3. $\left(G_{i,j} = G^{green}_{i,j}\right) \implies \left(u^{2j-i}S\left(1 - e^{-Y\Delta t}\right) \ge K \left(1 - e^{-R\Delta t}\right)\right)$

Proof. By $G_{i,j} = G^{green}_{i,j}$ and Def. 2.1, we know $S \cdot u^{2j-i} - K > s_0 G_{i+1,j} + s_1 G_{i+1,j+1}$, $G_{i+1,j} \ge S \cdot u^{2j-(i+1)} - K$, and $G_{i+1,j+1} \ge S \cdot u^{2(j+1)-(i+1)} - K$.

Denote $z = S \cdot u^{2j-i}$ and use above inequalities:

$$\begin{aligned}
z - K &\ge s_0(S\cdot u^{2j-(i+1)} - K) + s_1(S\cdot u^{2(j+1)-(i+1)} - K)\\
&= m(S\cdot u^{2j-(i+1)} - K) + mp((S\cdot u^{2(j+1)-(i+1)} - K)\\
&\quad - (S\cdot u^{2j-(i+1)} - K))\\
&= (1/u)\left(mz - mKu + mp\left(u^2-1\right)z\right)\\
&= \frac{1}{u}\left(mz - mKu + \frac{m(e^{(R-Y)\Delta t} - \frac{1}{u}}{u - \frac{1}{u}}(u^2-1)z\right)\\
&= \frac{1}{u}\left(-mKu + u\cdot e^{-Y\Delta t}z\right) = -e^{-R\Delta t}K + e^{-Y\Delta t}
\end{aligned}$$

Then we have $z(1-e^{-Y\Delta t}) \ge K(1-e^{-R\Delta t}) \implies u^{2j-i}S(1-e^{-Y\Delta t}) \ge K(1-e^{-R\Delta t})$. Thus, $\left(G_{i,j} = G^{green}_{i,j}\right) \implies \left(u^{2j-i}S(1-e^{-Y\Delta t}) \ge K(1-e^{-R\Delta t})\right)$ □

Lemma 2.4 shows that within the upper-left triangular area of G if a cell is green then the cell below it must also be green, and Lemma 2.6 shows that if a cell is red then the cell diagonally left below it must also be red.

Lemma 2.4. $\left(G_{i+1,j} = G^{green}_{i+1,j}\right) \implies \left(G_{i,j} = G^{green}_{i,j}\right)$ *for* $i \in [0, T-1]$ *and* $j \in [0,i]$.

Proof. $\left(G_{i+1,j} = G^{green}_{i+1,j}\right) \implies \left(G_{i+1,j+1} = G^{green}_{i+1,j+1}\right)$ by Lemma 2.2. Let $z = Su^{2j-i}$, then $G_{i+1,j} = \frac{z}{u} - K$ and $G_{i+1,j+1} = uz - K$.

$$G^{red}_{i,j} = s_0 G_{i+1,j} + s_1 G_{i+1,j+1} = m(1-p)\left(\frac{z}{u} - K\right) + mp(uz - K)$$

$$= -Su^{2j-i}\left(1 - e^{-Y\Delta t}\right) + Su^{2j-i} - Ke^{-R\Delta t}$$

$$\le -K\left(1 - e^{-R\Delta t}\right) + Su^{2j-i} - Ke^{-R\Delta t} \text{ (by Lemma 2.3 at } G_{i+1,j})$$

$$= Su^{2j-i} - K = G^{green}_{i,j}$$

Hence, the statement holds. □

Lemma 2.5. *For any* $i \in [0, T-2]$ *and* $j < i$: $G_{i,j} \ge G_{i+2,j+1}$.

Proof. We use mathematical induction. For $i = T-2$, we have our base case: $G_{i,j} \ge \max(0, Su^{2j-i} - K) = G_{T,j+1}$.

Suppose it holds true for all $G_{i+1,j}$ for some given $i \ge T-3$ and $j \in [0,i]$. Then $G_{i,j} = \max(s_0 G_{i+1,j} + s_1 G_{i+1,j+1}, Su^{2j-i} - K) \ge \max(s_0 G_{i+3,j+1} + s_1 G_{i+3,j+2}, Su^{2j-i} - K) = G_{i+2,j+1}$. □

Lemma 2.6. $\left(G_{i+1,j+1} = G^{red}_{i+1,j+1}\right) \Rightarrow \left(G_{i,j} = G^{red}_{i,j}\right)$ *for* $i \in [0, T-1]$ *and* $j \in [0, i-1]$.

Proof. $G_{i+1,j+1} = G^{red}_{i+1,j+1}$ and Lemma 2.4 gives: $G_{i+2,j+1} = G^{red}_{i+2,j+1} \ge G^{green}_{i+2,j+1} = Su^{2j-i} - K$. Now, by Lemma 2.5, $G_{i,j} \ge G_{i+2,j+1} \ge Su^{2j-i} - K = G^{green}_{i,j}$. So, $G_{i,j} = G^{red}_{i,j}$. □

The following corollary says that at every time step all red cells appear to the left of all green cells, and the boundary between the green and the red regions either remains the same or moves by one cell towards the left every time step.

Corollary 2.7. *For every $i \in [0, T-1]$, there exists an index $j_i \in [0, i]$ such that all cells $G_{i,j}$ with $0 \le j \le j_i$ are red and all (possibly zero) cells $G_{i,j}$ with $j_i < j \le i$ are green. Also, for $i \in [0, T-2]$, $j_{i+1} - 1 \le j_i \le j_{i+1}$.*

Proof. Follows from Lemmas 2.2, 2.4, and 2.6. □

2.3 Algorithm for American call option pricing under BOPM

The solution space is a right-angle isosceles triangle with base length T. We know the boundary between the red and green cells in the first row of the triangle (solution space); however, we do not know the locus of the boundary in the subsequent rows of the triangle. We compute the boundary in the following process.

We partition the triangle (solution space) into trapezoids (see Figure 3a). We compute the first trapezoid with its first row as the same first row of the triangle (the solution space) and solve this newly created trapezoid (we explain how we create a trapezoid and solve it later in this section). Then we compute the second trapezoid with its first row as the last row of the first trapezoid and solve the second trapezoid. This process continues until we are left with a right-angle isosceles triangle with base size at most $\sqrt{T}$. We solve this triangle iteratively by doing quadratic work in time $O(T)$. We describe the process in detail below.

Partitioning the Triangle into Trapezoids. Let abc be a right-angle isosceles triangle with base length T (see Figure 3a). Let ℓ_1 be the number of red cells on the line segment ab. They will appear consecutively from a to some point p. Let d be the point in the line segment ac that is ℓ_1 distance from a. Draw a horizontal line from d; let the line intersect bc at point e. Therefore, we get a trapezoid $abed$ with height ℓ_1 with ℓ_1 red cells in its first row.

We solve trapezoid $abed$, which means that we compute the values of all red cells in its last row and thus find the boundary point q between red and green cells in de. Let $|dq| = \ell_2$. Let f be the point on dc that $|df| = \ell_2$. We draw a horizontal line fg, and get the second trapezoid $degf$ of height ℓ_2. The last row of the trapezoid $abed$ becomes the first row of $degf$.

We solve the trapezoid $degf$ and all subsequent trapezoids created following the approach described above until we are left with a right-angle isosceles triangle xyc with base length at most $\sqrt{T}$. We compute all cell values of xyc iteratively in $O(T)$ time.

Solving a Trapezoid. Solving a trapezoid means given all red cell values in its first row computing all red cell values in its last row. Let $abcd$ be a trapezoid of height ℓ with ℓ red cells from a to q in its first row (Figure 3b). Let r be the point on ad such that $|ar| = \lfloor \ell/2 \rfloor$. Draw a horizontal line from r intersecting bc at v. To compute the values of red cells on dc, we (1) compute all red cells on rv, and (2) using the cell values on rv, compute all red cells on dc.

(1) *Computing all red cells on* $\boldsymbol{rv}$. Let rv and qd intersect t. We compute the cell values on line segment rt using the FFT-based stencil algorithm of [2] which are guaranteed to be red because of Corollary 2.7. Note that $|rt| = |pt| = \lfloor \ell/2 \rfloor$. The newly created trapezoid $pbvt$ has height $\lfloor \ell/2 \rfloor$, and there are $\lfloor \ell/2 \rfloor$ red cells in its first row. We solve trapezoid $pbvt$ recursively similar to trapezoid $abcd$, and thus compute all red cell values on tv.

(2) *Computing all red cells on* $\boldsymbol{dc}$ *using the red cells on* $\boldsymbol{rv}$. Let point u be on the boundary between red and green cells on rv. Draw a line from u parallel to qd intersecting dc at w. Next, draw a line through w perpendicular to dc intersecting rv at point s. We compute all red cells on dc in exactly the same way we computed the red cells on rv above. We first use the FFT-based stencil algorithm of [2] to find all cell values on dw, and then recursively solve trapezoid $svcw$ of height $\lceil \ell/2 \rceil$ to find all red cell values on wc.

Solving trapezoids of height $\boldsymbol{O(1)}$ *(base case).* We solve trapezoids of height $O(1)$ in $O(1)$ work and span using the naïve looping code (e.g., see the pseudocode in Figure 1).

The following theorem gives the work and span of our algorithm.

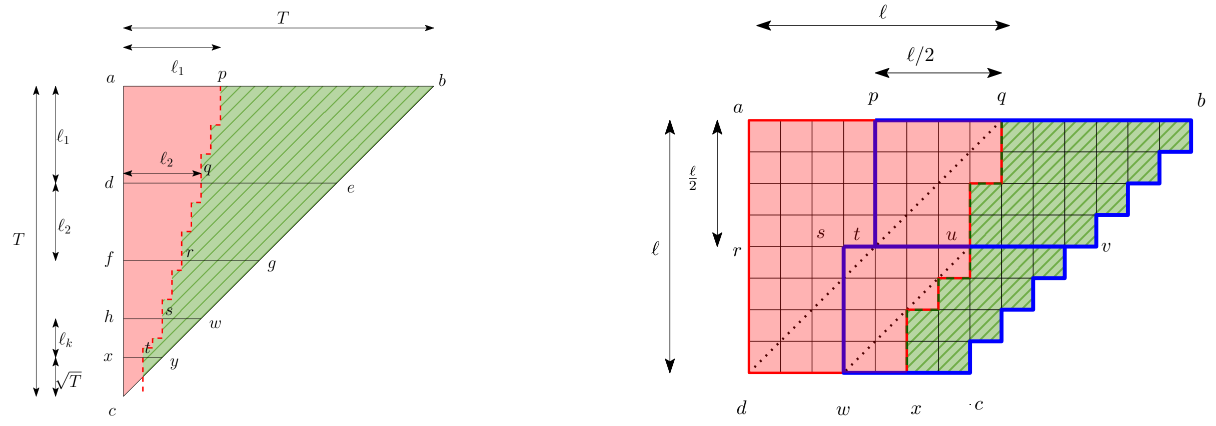
Theorem 2.8. *Our algorithm solves the American call option pricing problem under BOPM in $O(T \log^2 T)$ work and $O(T)$ span, where T is the number of time steps.*

Proof. Let $\zeta_1(\ell)$ and $\zeta_\infty(\ell)$ be the work and span, respectively, for solving a trapezoid of height ℓ recursively. We call the $O(\ell \log \ell)$-work FFT-based periodic algorithm [2] twice and solve two smaller trapezoids of height $\ell/2$ each recursively. Hence, $\zeta_1(\ell) = 2\zeta_1(\lceil \ell/2 \rceil) + \Theta(\ell \log \ell) = O(\ell \log^2 \ell)$. Observe that although the two smaller trapezoids must be solved one after the other (e.g., $svcw$ after $pbvt$ in Figure 3b), each of them can be solved in parallel with the FFT-based algorithm (of span $O(\log \ell \log \log \ell)$ [2]) called to find the red cells on its left (on line segments rt and dw, respectively). Hence, $\zeta_\infty(\ell) = 2\zeta_\infty(\lceil \ell/2 \rceil) + O(\log \ell \log \log \ell) = O(\ell)$.

Suppose that we solve k trapezoids of heights $\ell_1, \ell_2, \ldots, \ell_k$, respectively, using the above process as shown in Figure 3a. Let Ψ_1 and Ψ_∞ be the total work and span, respectively, for solving those k trapezoids followed by the time needed to solve the leftover triangle of height $O\left(\sqrt{T}\right)$. Then $\Psi_1 = \left(\sum_{1 \le i \le k} O\left(\ell_i \log^2 \ell_i\right)\right) + O(T) = O\left(T \log^2 T\right)$. Since those trapezoids and the triangle are solved in sequence, we have $\Psi_\infty = \sum_{1 \le i \le k} O(\ell_i) + O\left(\sqrt{T}\right) = O(T)$. □

3 American Call Option under the Trinomial Option Pricing Model

The trinomial options pricing model (TOPM) encodes the possible sequences of prices for a given asset within the structure of a trinomial tree (see Figure 2(c)). It expands on BOPM by allowing the value of an asset to remain unchanged



(a) Partitioning the solution space into trapezoids **(b)** Decomposing a trapezoid into smaller trapezoids

Figure 3. American call option pricing under BOPM

after a given time step. TOPM was introduced by Boyle [21], and while it is less popular than the BOPM, it provides for the possibility of more accurate predictions than BOPM at the cost of only a constant factor blowup in runtime when using naïve $O\left(T^2\right)$ methods. Langat, Mwaniki, and Kiprop showed that TOPM converges to the same solution as Black-Scholes with half as many time steps [63]. TOPM is also equivalent to the explicit finite difference method [52].

TOPM carries over many properties from BOPM, e.g., $X_{node} = S \times u^{N_u - N_d}$ since the number of "remain the same" moves does not factor into the price. Here $u = e^{V\sqrt{2\Delta t}}$, and $d = 1/u$. The exercise value in TOPM is $\max(X_{node} - K, 0)$.

The transition probabilities can be expressed as $p_u = \left(\frac{(e^{(R-Y)\Delta t/2} - \sqrt{d})}{(\sqrt{u} - \sqrt{d})}\right)^2$, $p_d = \left(\frac{(\sqrt{u} - e^{(R-Y)\Delta t/2})}{(\sqrt{u} - \sqrt{d})}\right)^2$, and $p_o = 1 - p_u - p_d$, which are alternate forms of those given in [52].

Let $m = e^{-R\Delta t}$, $s_0 = mp_u, s_1 = mp_o, s_2 = mp_d$. Let $G_{i,j}$ denote the grid value in the row $i \in [0, T]$ and column $j \in [0, 2i]$ of the $(T+1) \times (2T+1)$ grid G. Let $G_{i,j}^{green} = S \cdot u^{j-i} - K$, and let $G_{i,j}^{red} = \sum_{k \in \{0,1,2\}} s_k G_{i+1,j+k}$ if $i \in [0, T)$, and 0 otherwise. Then similar to BOPM:

$$G_{i,j} = \begin{cases} G_{i,j}^{red}, & \text{if } G_{i,j}^{red} \geq G_{i,j}^{green} \\ G_{i,j}^{green}, & \text{otherwise.} \end{cases}$$

In the full version of the paper [4], we show that the TOPM grid shows properties similar to the BOPM grid, that is, in every row all red cells appear first in contiguous locations followed by all green cells, and with every time step the red-green boundary moves by at most one cell to the left. This allows us to use a similar algorithm to that given for BOPM (Section 2.3) with the identical work and span.

4 American Put Option under the Black-Scholes-Merton Model

4.1 Black-Scholes-Merton Pricing Model (BSM)

BSM is a mathematical method to calculate the theoretical value of an option contract. The option pricing problem is transformed into a partial differential equation (PDE) with variable coefficients. An explicit formula for the price can be obtained assuming a log-normal distribution of the asset price. Note that the limit of the discrete-time BOPM approximates the continuous-time BSM under the same assumption. While BOPM utilizes simple statistical methods, BSM requires a solution of a stochastic differential equation.

Denote stock price at time t by $\mathcal{S}(t)$. BSM claims that there is a deterministic relation between the option price and the stock price and time. This means that there is a deterministic function $v(t,x)$ for option price x and time t such that: $X(t) = v(t, \mathcal{S}(t))$, where $X(t)$ represents the value of the option at time t. Now BSM derives that $v(t,x)$ satisfies the following two-area classical form:

$$v(t,x) = \begin{cases} \frac{1}{r}\left(\begin{array}{l}\frac{\partial v}{\partial t}(t,x) + rx\frac{\partial v}{\partial x}(t,x) \\ +\frac{1}{2}\sigma^2 x^2 \frac{\partial^2 v}{\partial x^2}(t,x)\end{array}\right), & \text{if } x > L(T-t) \\ K - x, & \text{if } 0 \leq x \leq L(T-t) \end{cases} \quad (1)$$

where $L(0) = K$, $\frac{\partial v}{\partial x}(t, L(T-t)^+) = \frac{\partial v}{\partial x}(t, L(T-t)^-) = -1$, and $v(t, L(T-t)^+) = v(t, L(T-t)^-)$ for $0 \leq t < T$.

It is equivalent to satisfying the following:

$$v(t,x) \geq \begin{cases} \frac{1}{r}\left(\begin{array}{l}\frac{\partial v}{\partial t}(t,x) + rx\frac{\partial v}{\partial x}(t,x) \\ +\frac{1}{2}\sigma^2 x^2 \frac{\partial^2 v}{\partial x^2}(t,x)\end{array}\right), & \text{if } 0 \leq x \leq L(T-t) \\ K - x, & \text{if } x > L(T-t) \end{cases} \quad (2)$$

where, $t \in [0, T]$, Recall that at the maturity time T, the option price (call option case) will be $X(T) = \max(\mathcal{S}(T) - K, 0) = (K - \mathcal{S}(T))^+$.

It also means that $v(T,x) = (K-x)^+$ on the boundary.

Therefore, the goal becomes solving Equation (1) or Inequality (2). For more details on how the BSM model is formulated or why the complementary form is equivalent to the classical form, see [26, 100, 112]. We show the two areas are contiguous and find properties of their boundary in Section 4.2, which can be exploited by our algorithm in Section 4.3.

4.2 Properties of the Two-Area Boundary

Note that Equation (1) includes dimensional variables. First, we find nondimensionalized forms of Equations (1) and (2).

Let $s = \ln\frac{x}{K}$, $\tau = \frac{1}{2}\sigma^2(T-t)$, $\tilde{v}(\tau,s) = \frac{1}{K}v(t,x)$, $\tilde{L}(\tau) = L(T-t)$ and $\omega = \frac{2r}{\sigma^2}$. Then: $\frac{\partial\tilde{v}}{\partial s} = \frac{x}{K}\frac{\partial v}{\partial x}$, $\frac{\partial^2\tilde{v}}{\partial s^2} = \frac{x^2}{K}\frac{\partial^2 v}{\partial x^2} + \frac{x}{K}\frac{\partial v}{\partial x}$, $\frac{\partial\tilde{v}}{\partial\tau} = -\frac{2}{K\sigma^2}\frac{\partial v}{\partial t}$.

Applying Equation (1), and $\tilde{v}(\tau,s)$ satisfes the following:

$$\tilde{v}(\tau,s) = \begin{cases} \frac{1}{\omega}\begin{pmatrix}(\omega-1)\frac{\partial\tilde{v}}{\partial s}(\tau,s)\\ +\frac{\partial^2\tilde{v}}{\partial s^2}(\tau,s) - \frac{\partial\tilde{v}}{\partial\tau}(\tau,s)\end{pmatrix}, & \text{if } s > \tilde{L}(\tau) \\ 1-e^s, & \text{if } s \le \tilde{L}(\tau)\end{cases} \quad (3)$$

It also satisfies the dimensionless complementary form after plugging into Inequality (2):

$$\tilde{v}(\tau,s) \ge \begin{cases} \frac{1}{\omega}\begin{pmatrix}(\omega-1)\frac{\partial\tilde{v}}{\partial s}(\tau,s)\\ +\frac{\partial^2\tilde{v}}{\partial s^2}(\tau,s) - \frac{\partial\tilde{v}}{\partial\tau}(\tau,s)\end{pmatrix}, & \text{if } s \le \tilde{L}(\tau) \\ 1-e^s, & \text{if } s > \tilde{L}(\tau)\end{cases} \quad (4)$$

where, $\tilde{L}(0) = 1$ and $\tilde{v}(s,0) = \max(1-e^s, 0)$. Consider an approximation v_k^n of $\tilde{v}(n\Delta\tau, k\Delta s)$ and use the following finite-difference approximations (where, $\tilde{v}$ denotes $\tilde{v}(n\Delta\tau, k\Delta s)$):

$$\frac{\partial\tilde{v}}{\partial\tau} \approx \frac{v_k^{n+1} - v_k^n}{\Delta t}, \frac{\partial\tilde{v}}{\partial s} \approx \frac{v_{k+1}^n - v_{k-1}^n}{2\Delta s}, \frac{\partial^2\tilde{v}}{\partial s^2} \approx \frac{v_{k+1}^n - 2v_k^n + v_{k-1}^n}{(\Delta s)^2}$$

Now plug it into Equation (3) to obtain these two regions:

$$v_k^{n+1} = \begin{cases} \begin{pmatrix} \left(1-\omega\Delta\tau - 2\frac{\Delta\tau}{(\Delta s)^2}\right)v_k^n + \\ \sum\limits_{h\in\{-1,1\}}\left(\frac{\Delta\tau}{(\Delta s)^2} + h\frac{(\omega-1)}{2}\frac{\Delta\tau}{\Delta s}\right)v_{k+h}^n \end{pmatrix}, & \text{if } k > \widehat{L} \\ 1-e^{k\Delta s}, & \text{if } k \le \widehat{L}\end{cases} \quad (5)$$

where, $\widehat{L} = \frac{\tilde{L}((n+1)\Delta\tau)}{\Delta s}$ and $v_k^0 = \max(1-e^{k\Delta s}, 0)$ for all integer k. It also satisfies the following condition by discretizing Equation (4):

$$v_k^{n+1} \ge \begin{cases} \begin{pmatrix} \left(1-\omega\Delta\tau - \frac{2\Delta\tau}{(\Delta s)^2}\right)v_k^n + \\ \sum\limits_{h\in\{-1,1\}}\left(\frac{\Delta\tau}{(\Delta s)^2} + \frac{h(\omega-1)}{2}\frac{\Delta\tau}{\Delta s}\right)v_{k+h}^n \end{pmatrix}, & \text{if } k \le \widehat{L} \\ 1-e^{k\Delta s}, & \text{if } k > \widehat{L}\end{cases} \quad (6)$$

Similar to the BOPM for American option, we define the green zone and the red zone:

Definition 4.1. We call that v_k^n is in the green zone when $k\Delta s \le \tilde{L}(n\Delta\tau)$, otherwise it is in the red zone.

To apply Equations (5)–(6), we need to determine the form of $\tilde{L}((n+1)\Delta\tau)$ which has no analytical form, although it has asymptotic results [5] or approximation results [127, 128]. Instead we can use the following theorem from [26]:

Theorem 4.2 ([26]). *The early exercise boundary curve $\tilde{L}(\tau)$ is monotonically decreasing.*

Theorem 4.3. *Let $a = \frac{\Delta\tau}{(\Delta s)^2} + (\omega-1)\frac{\Delta\tau}{\Delta s}$, $b = \frac{\Delta\tau}{(\Delta s)^2} - (\omega-1)\frac{\Delta\tau}{\Delta s}$, $c = 1-a-b-\omega\Delta\tau$, and k_n be the largest integer such that $k_n \le \frac{\tilde{L}(n\Delta\tau)}{\Delta s}$. Then $0 \le k_n - k_{n+1} \le 1$ when $a, b, c \ge 0$.*

Proof. We first prove that $k_n - k_{n+1} \ge 0$. Suppose that this is not true. Then we will have: $\tilde{L}((n+1)\Delta\tau) \ge k_{n+1} \ge k_n + 1 > \tilde{L}(n\Delta\tau)$, which contradicts Theorem 4.2.

Now we prove that $k_n - k_{n+1} \le 1$. Because k_n is in the green zone, we will have the following:

$$v_{k_n}^n = 1 - e^{k_n\Delta s} \ge (1-\omega\Delta\tau - a - b)v_{k_n}^{n-1} + av_{k_n+1}^{n-1} + bv_{k_n-1}^{n-1}$$
$$\ge (1-\omega\Delta\tau-a-b)\left(1-e^{k_n\Delta s}\right) + a\left(1-e^{(k_n+1)\Delta s}\right)$$
$$+ b\left(1-e^{(k_n-1)\Delta s}\right)$$
$$\Rightarrow \omega\Delta\tau\left(1-e^{k_n\Delta s}\right) + e^{k_n\Delta s}\left(a\left(e^{\Delta s}-1\right) + b\left(e^{-\Delta s}-1\right)\right) \ge 0$$
$$\Rightarrow 1-e^{(k_n-1)\Delta s} > (1-\omega\Delta\tau-a-b)\left(1-e^{(k_n-1)\Delta s}\right)$$
$$+ a\left(1-e^{(k_n)\Delta s}\right) + b\left(1-e^{(k_n-2)\Delta s}\right)$$

Considering $v_{k_n-1}^{n+1}$, we first observe that:

$$v_{k_n-2}^n = 1-e^{(k_n-2)\Delta s}, v_{k_n-1}^n = 1-e^{(k_n-1)\Delta s}, v_{k_n}^n = 1-e^{(k_n)\Delta s}$$

Now we show that $v_{k_n-1}^{n+1}$ is in the green zone. Suppose that it is in the red zone. Then:

$$v_{k_n-1}^{n+1} = (1-\omega\Delta\tau-a-b)\left(1-e^{(k_n-1)\Delta s}\right) + a\left(1-e^{(k_n)\Delta s}\right)$$
$$+ b\left(1-e^{(k_n-2)\Delta s}\right) < 1-e^{(k_n-1)\Delta s}$$

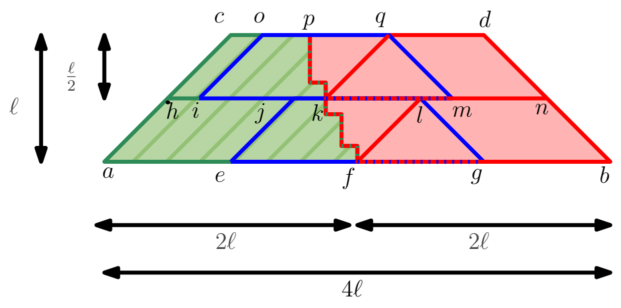
which leads to a contradiction because $v_{k_n-1}^{n+1}$ should be $\ge 1-e^{(k_n-1)\Delta s}$. By the definition of k_{n+1}, we must have $k_{n+1} \ge k_n - 1$, completing the proof of the theorem. □

In this theorem, we require $a, b, c \ge 0$ which is necessary for stability of the finite difference scheme. The notion of stability in numerical differential equations is a crucial aspect of numerical analysis. It is about ensuring that the numerical solution of a differential equation behaves in a consistent and predictable manner, especially over long time intervals or fine spatial resolutions. The stability condition is related to the CFL (Courant-Friedrichs-Lewy) condition [110] for wave equations.

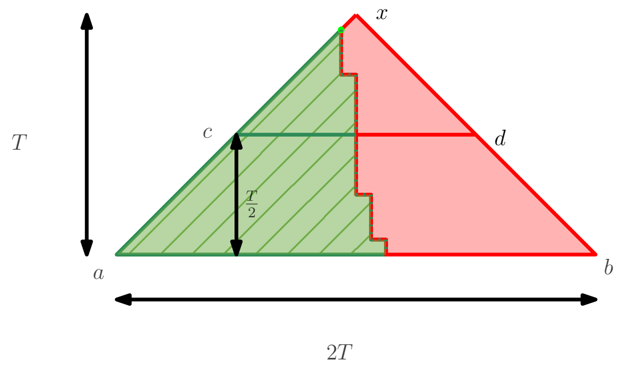
4.3 Algorithm for American put option pricing under BSM

Our algorithm for the American put option under BSM is similar to our algorithm for the American call option under BOPM as described in Section 2.

Observe that we will have to perform a nonlinear stencil computation based on the update equation (5). For T time steps we use a $T\times 2T$ space-time grid with the time dimension being T and spatial dimension $2T$. According to Equation (5), we compute a cell v_k^{n+1} of that grid, where $n+1$ represents the time coordinate and k the spatial coordinate, from cells v_{k-1}^n,



(a) Decomposing trapezoid *abcd* into smaller trapezoids



(b) Partitioning the solution space into trapezoids

Figure 4. American put option pricing under BSM

v_k^n, and v_{k+1}^n using a 3-point stencil provided $k > \frac{\tilde{L}((n+1)\Delta\tau)}{\Delta s}$. Otherwise, we set it to $1 - e^{k\Delta s}$. In the first case, cell v_k^{n+1} will be in the red zone, and in the second case it will be in the green zone. As explained in Section 4.2, the entire boundary between these two zones in not known ahead of time, but it moves by at most one cell toward the green region with every time step. The goal of the algorithm is to compute the value of the central cell of the spatial dimension at time step T (e.g., apex x of the isosceles triangle abx in Figure 4b).

We solve the problem by decomposing the isosceles triangle abx into a sequence of the isosceles trapezoids of geometrically decreasing heights (see Figure 4b) and solving (i.e., find the cell values of top base given those of the bottom base of the trapezoid) them one by one from bottom to top until we reach a leftover triangle of small constant size which we solve naïvely to find the value of x. We solve an isosceles trapezoid recursively by decomposing it into two smaller trapezoids of smaller height and solving them recursively and also using the FFT-based algorithm from [2] solve two subtrapezoids that are entirely composed of red cells (see Figure 4a). Details of this algorithm are given below.

We first show how to solve an isosceles trapezoid $abdc$ (as shown in Figure 4a) of height l, bottom/longer base (ab) length 4ℓ, and $\angle cab = \angle dba = 45°$. Thus, the top/shorter base dc is of length 2ℓ. Solving trapezoid $abdc$ means computing the values of the cell at the top base cd given the values of the cells at the bottom base ab.

If $\ell \le 10$, we naïvely solve $abdc$ and identify the location of the red-green boundary point p in cd in $O(1)$ time. If $\ell > 10$, we find the row hn at height $\frac{\ell}{2}$ and calculate all the cells in it. To do so, we recursively solve the trapezoid $eglj$ which is found as follows:

1. Let f be the point on ab that lies in the green region, but $f+1$ is in the red region. Identify the points e and j to the left and right of f, respectively, such that $|ef| = |fj| = \ell$.

2. Construct an isosceles trapezoid with base eg, height $\frac{\ell}{2}$ and top jl such that $\angle jeg = \angle lge = 45°$. Thus, $|jl| = \ell$.

After solving the trapezoid $eglj$, we have the cell values in jl and the location of the red-green boundary point k in jl. We can easily calculate the values of cells in hj since those values are independent of time and depend only on spatial coordinates. Finally, we use the *FFT*-based algorithm of [2] to solve the trapezoid $fbnl$, where the point l is found by forcing $fbnl$ to be an isosceles trapezoid.

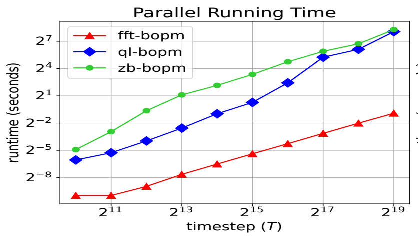
Therefore, we can get the values of the cells in ln and thus the values of all the cells in hn. Then we can calculate the values of the cells in dc given the values of the cells in hn exactly the same way as we computed the cell values in hn from those in ab.

Now, let us go back to Figure 4b to see how to compute the value of the apex x. After solving $abdc$ as above to calculate the cells in cd, if $|cd| \le 10$, we naïvely calculate the value of x, which takes $O(1)$ time. But if $|cd| > 10$, we recursively apply our trapezoid algorithm to solve a smaller trapezoid with the bottom base cd.

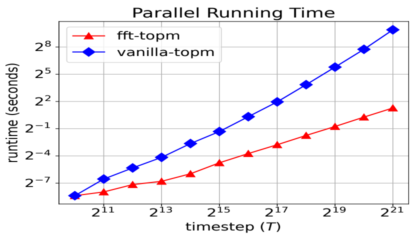
Theorem 4.4. *Our algorithm solves the American put option pricing problem under BSM in $O(T\log^2 T)$ work and $O(T)$ span, where T is the number of time steps.*

Proof. The proof is very similar to that of Theorem 2.8. Let $\zeta_1(\ell)$ and $\zeta_\infty(\ell)$ be the work and span, respectively, for solving a trapezoid of height ℓ (see Figure 4a). We recursively solve two trapezoids of height $\ell/2$ each in sequence but use a parallel FFT-based algorithm [2] on each half, both size $\Theta(\ell)$. Since the FFT-based algorithm performs $O(\ell\log\ell)$ work in $O(\log\ell\log\log\ell)$ span, we can write: $\zeta_1(\ell) = 2\zeta_1\left(\frac{\ell}{2}\right) + O(\ell\log\ell)$ if $\ell > 10$ and $O(1)$ otherwise. Similarly, $\zeta_\infty(\ell) = 2\zeta_\infty\left(\frac{\ell}{2}\right) + O(\log\ell\log\log\ell)$ if $\ell > 10$ and $O(1)$ otherwise. Solving, $\zeta_1(\ell) = O\left(\ell\log^2\ell\right)$ and $\zeta_\infty(\ell) = O(\ell)$.

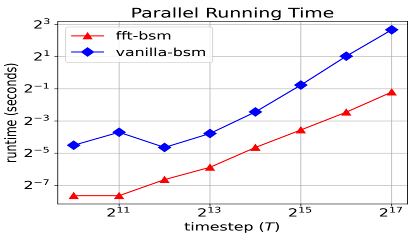
Now, let $\Psi_1(T)$ and $\Psi_\infty(T)$ be the work and the span, respectively, of solving an isosceles triangle of base size T (see Figure 4b). Then $\Psi_1(T) = \Psi_1\left(\frac{T}{2}\right) + O\left(T\log^2 T\right)$ if $T > 10$ and $O(1)$ otherwise. Also, $\Psi_\infty(T) = \Psi_\infty\left(\frac{T}{2}\right) + O(T)$ if $T > 10$ and $O(1)$ otherwise. Solving, $\Psi_1(T) = O\left(T\log^2 T\right)$ and $\Psi_\infty = O(T)$. □



(a) BOPM

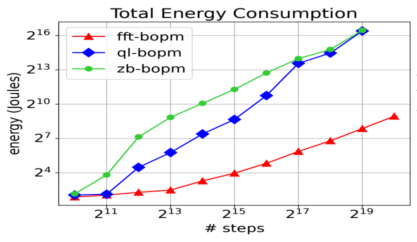


(b) TOPM

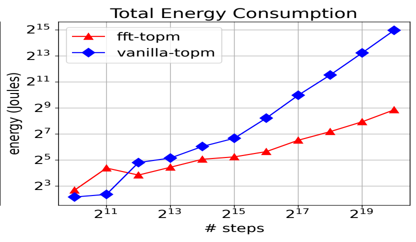


(c) BSM

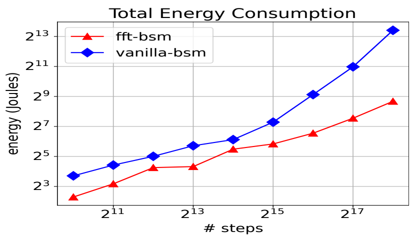
Figure 5. Running time comparisons of our algorithms with state-of-the-art parallel algorithms.



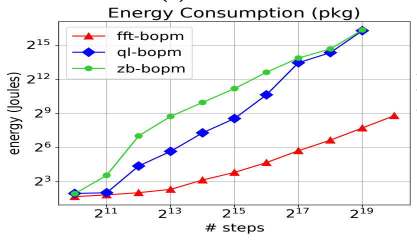
(a) BOPM



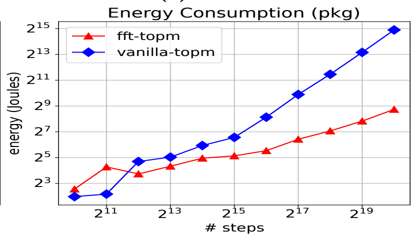
(b) TOPM



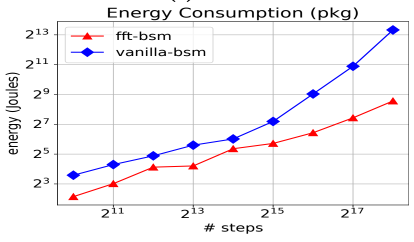
(c) BSM



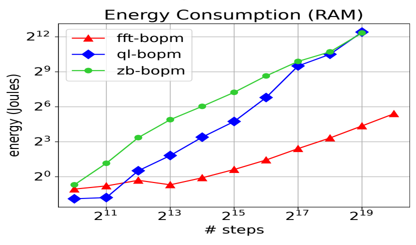
(d) BOPM



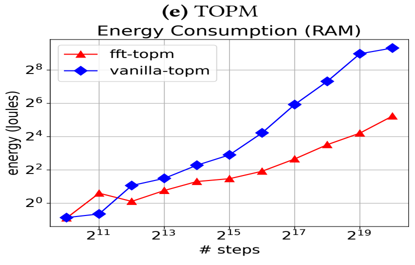
(e) TOPM



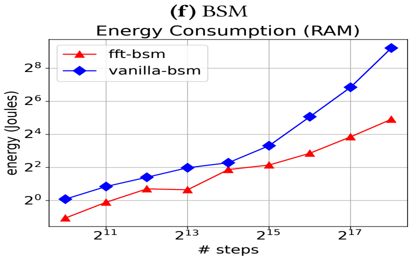
(f) BSM



(g) BOPM

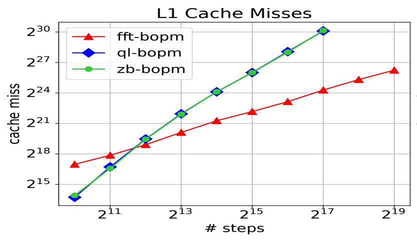


(h) TOPM

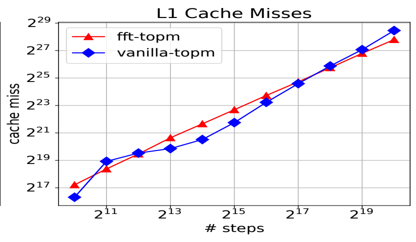


(i) BSM

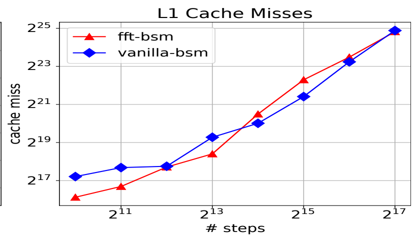
Figure 6. Comparison of energy consumption: total ((a)–(c)), package ((d)–(f)), and RAM ((h)–(i)).



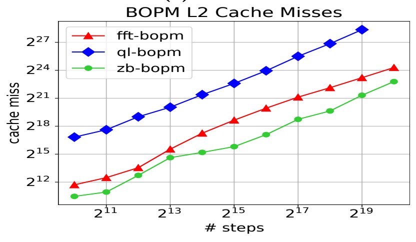
(a) BOPM



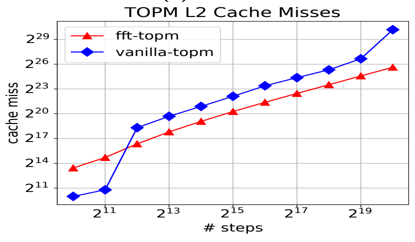
(b) TOPM



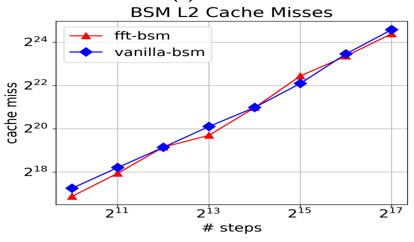
(c) BSM



(d) BOPM



(e) TOPM



(f) BSM

Figure 7. Comparison of cache misses: L1 ((a)–(c)) and L2 ((d)–(f)).

Table 3. Experimental setup on a Stampede2 [1] SKX node.

Processor	Intel Xeon Platinum 8160 (Skylake / SKX)
Cores	24 cores per socket, 2 sockets (total: 48 cores)
Cache sizes	L1 32 KB / core, L2 1 MB / core, L3 33 MB / socket
Memory	144GB /tmp partition on a 200GB SSD
Compiler	Intel C++ Compiler (ICC) v18.0.2
Compiler flags	`-O3 -xhost -ansi-alias -ipo -AVX512`
Parallelization	OpenMP 5.0
Thread affinity	`GOMP_CPU_AFFINITY`

Table 4. Legends used in plots and tables.

Legend	Meaning
`fft-bopm`, `fft-topm`, `fft-bsm`	our FFT-based implementations for BOPM, TOPM, and BSM, respectively
`ql-bopm`, `zb-bopm`	BOPM implementations from Par-bin-ops based on QuantLib and Zubaer et al.'s work, respectively
`vanilla-topm`, `vanilla-bsm`	our parallel looping implementations for TOPM and BSM, respectively

5 Experimental Results

In this section, we present an experimental evaluation of our algorithms and compare them with the fastest existing solutions. Our experimental setup is shown in Table 3. The legends used in our plots are listed in Table 4, which are described in more detail in the next few paragraphs.

Benchmarks. For benchmarks, we use American call option pricing under BOPM and TOPM, and American put option under the BSM. For the BOPM call option benchmarks, our baselines are the option call probability calculations from QuantLib [29] and Zubair et al.'s parallel cache optimized model [130]. We use the optimized implementations of these two baselines available in Par-bin-ops [23]. These implementations are the fastest existing implementations of BOPM call option pricing. For the TOPM call option and BSM put option, our FFT-based implementations are compared with our parallel looping-based vanilla implementations, as we could not find any publicly available faster implementations.

We use the *perf* (version: 3.10.0-1160.53.1.el7.x86_64.debug) tool [84] to analyze the system-wide energy consumption, and the PAPI (version: 5.6.0) library [117] for cache miss counts for our implementations and benchmarks.

Par-bin-ops. In 2022, Brunelle et al. [23] released an open-source framework that can leverage parallel, cache-optimized algorithms to compute a variety of binomial option types and enables a simple interface for developers. We used the latest version from Github. Experimental evaluations by Brunelle et al. [23] has shown that Par-bin-ops achieves more than 139× speedup over the QuantLib library when evaluating a European call option using 200,000 steps. Therefore, we chose the Par-bin-ops tool to benchmark our implementations of our FFT-based algorithms. To have a valid comparison of running times, we use Par-bin-ops for both QuantLib and Zubaer et al.'s [130] option probability calculation equations and report the running times comparing with our FFT-based implementations. We use the stencil-based cache-optimized version of Zubaer et al.'s algorithm from Par-bin-ops.

Parameter Values. As Table 2 shows, the only option pricing parameter that influences the performance bounds of the algorithms in our experiments is the number of time steps T. Therefore, we keep all other option pricing parameters fixed in all of our experiments. We use the following parameter values: $E = 252$, $K = 130$, $S = 127.62$, $R = 0.00163$, $V = 0.2$, $Y = 0.0163$ (notations are in Table 1).

5.1 Parallel Running Times

Figure 5(a) shows the parallel running time comparison of American call option pricing calculations under BOPM. Our FFT-based algorithm uses a recursive divide-and-conquer approach. We have found empirically that a base case size of 8 steps yields the best running times. Experimental results show that our FFT-based algorithm can outperform Par-bin-ops for any number of step sizes for both serial and parallel implementations. We achieve more than 16× speedup for $T \approx 1000$ and more than 500× speedup for $T \approx 0.5$ million w.r.t. Par-bin-ops implementations.

Our TOPM algorithm runs more than 2.5× faster for $T \approx 2000$ and more than 390× faster for $T \approx 2.1$ million w.r.t. the parallel vanilla code. Figure 5(b) shows the comparisons.

Figure 5(c) shows the parallel running time comparisons of the American put option pricing computations under BSM. Our FFT-based implementation is compared with the looping-based vanilla implementation. Our algorithm achieves more than 8× speedup for $T \approx 1000$ and more than 14× speedup for $T \approx 0.13$ million w.r.t. the vanilla implementation.

5.2 Energy Consumption

Figure 6 shows the comparison of system-wide energy consumption while running our FFT-based implementation and respective benchmarks. We collected the energy consumption estimate of the entire package (*pkg*) and the main memory (*RAM*) through the RAPL (Running Average Power Limit) interface of model-specific registers (MSR) using the *perf* [84] profiling tool. Our FFT-based BOPM, TOPM, and BSM implementations consume 99%, 99%, and 96% less energy, respectively, compared to their benchmarks for large T values used in our experiments. For $T \approx 4000$, the energy savings are 80%, 50%, and 40%, respectively.

5.3 Cache Misses

Figure 7 shows L1 cache-miss (= L2 cache access) and L2 cache-miss results, respectively, for all implementations. Our FFT-based implementation incurs far fewer L1 cache misses than both Par-bin-ops implementations for BOPM. However, while we incur far fewer L2 cache-misses than `ql-bopm`, the other one (`zb-bopm`) incurs fewer L2 misses than ours. In case of TOPM, while our FFT-based implementation incurs far fewer L2 misses than our parallel looping implementation (`vanilla-topm`), the trend is not clear for L1 misses. For BSM, neither L1 nor L2 misses seem to have a clear winner.

5.4 Scalability

As Table 2 shows, our algorithms reduce the span $\mathcal{T}_\infty$ of solving the option pricing problems that we consider by a factor of $\Omega(\log T)$, but they reduce the work $\mathcal{T}_1$ by a substantially larger factor of $\Theta\left(T/\log^2 T\right)$. While the reduction in work leads to a significant reduction in energy consumption (see Section 5.2), it also leads to a low parallelism of $\mathcal{T}_1/\mathcal{T}_\infty = \Theta\left(\log^2 T\right)$. But it is still easy to see from the complexities given in Table 2 that the parallel running time $\mathcal{T}_p$ of our algorithms will be asymptotically lower than that of the corresponding fastest existing parallel algorithms for every value of p (stated in Proposition 1.1).

Table 5. Parallel run times (in ms) for $T = 2^{15}$ as p varies.

	$p=1$	$p=2$	$p=4$	$p=8$	$p=16$	$p=32$	$p=48$
`fft-bopm`	32	28	24	26	29	33	38
`ql-bopm`	26552	12498	6785	3530	1950	1324	1191

Table 5 above shows how the parallel running times of our FFT-based BOPM implementation (`fft-bopm`) and that of the QuantLib-based implementation from Par-bin-ops (`ql-bopm`) vary with p as T is kept fixed at 2^{15}. We see that although the parallel running time of `fft-bopm` stops decreasing somewhere between $p = 4$ and $p = 8$, it remains significantly faster than `ql-bopm` even when $p = 48$. Our algorithm scales better as T increases, e.g., for $T = 2^{19}$, it scales to a $p \in [8, 16)$, and compared to a subsecond running time of `fft-bopm` for $p = 1$, `ql-bopm` takes $\approx$ 2 hours to run.

6 Applications to Other Problems

Our model can be extended to encompass general finance problems, particularly those under the regime-switching framework. Beyond the realm of finance, our algorithms prove to be effective for physical models, like heat conduction in a rod with varying materials or the one-dimensional wave equation in a rod composed of two distinct materials. The crux of our model's utility lies in its adaptability to stencil computations, wherein different regions follow distinct stencil update rules. Now we will present several examples that our algorithm can apply to.

Example 6.1 (Heat equation with two medium)**.** Let α_1 and $u_1(x, y, t)$ be the thermal diffusivity and temperature distribution, respectively, of the first medium. Let α_2 and $u_2(x, y, t)$ be the same for the second medium. Then the heat equations will be as follows:

$$\frac{\partial u_1}{\partial t} = \alpha_1 \nabla u_1 \quad \text{and} \quad \frac{\partial u_2}{\partial t} = \alpha_2 \nabla u_2$$

At the interface between the two media $\partial\Omega \times [0, T]$, we will have a continuity of temperature and heat flux when $(x, y, t) \in \partial\Omega \times [0, T]$:

$$u_1(x, y, t) = u_2(x, y, t) \quad \text{and} \quad \frac{\partial u_1(x, y, t)}{\partial \hat{\mathbf{n}}} = \frac{\partial u_2(x, y, t)}{\partial \hat{\mathbf{n}}},$$

where $\hat{\mathbf{n}}$ denotes normal to the $\partial\Omega$ pointing outward direction. Now the u on the whole region will be:

$$u(x, y, t) = \begin{cases} u_1(x, y, t), \text{ if } (x, y, t) \text{ is on the left of } \partial\Omega \times [0, T] \\ u_2(x, y, t), \quad \text{otherwise} \end{cases}$$

The function u will also need to satisfy some initial condition f at $t = 0$: $u(x, y, 0) = f(x, y)$.

Example 6.2 (Heat equation with non uniform stencil computation)**.** Let us consider a simple linear heat equation $u_t = \alpha u_{xx}$ with initial condition $u(x, 0) = e^{-100(x-0.5)^2}$ and boundary condition $u(0, t) = u(1, t) = 0$.

Observe that the function $y = e^{-100(x-0.5)^2}$ will drop sharply away from $x = 0.5$ in both directions and become very flat when $|x - 0.5|$ is large. As a result, it will be beneficial to have a smaller step size near 0.5 and a larger step size when x is far away from 0.5, which leads to multi-media updates.

7 Conclusion and Future Work

We have designed fast American option pricing algorithms under the binomial, trinomial, and the Black-Scholes-Merton models. We solve a type of nonlinear stencil problem that is of independent interest with potential applications beyond quantitative finance.

Future work may explore other models for American option pricing, such as the time dependent volatility model, stochastic volatility model, and the jump diffusion model. European and Asian options, Lookback options, Knock-out Barrier options, and Bermudan options are also of interest.

How to extend Ahmad et al.'s linear stencil algorithms based on Gaussian approximations [3] for finding fast approximate solutions to American option pricing and other classes of nonlinear stencil problems is also worth exploring.

Acknowledgements. This research was partly funded by NSF grant CCF-2318633 and an SBU OVPR seed grant (award #93214). We also thank Julian Shun, Yuanhao Wei, Shangdi Yu, Pranali Vani, and Sabiyyah Ali for valuable discussions and a careful review of the paper, which helped enhance its quality and ensure correctness.

References

[1] Stampede2. The Stampede2 supercomputing cluster. https://www.tacc.utexas.edu/systems/stampede2.

[2] Zafar Ahmad, Rezaul Chowdhury, Rathish Das, Pramod Ganapathi, Aaron Gregory, and Yimin Zhu. 2021. Fast stencil computations using fast Fourier transforms. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures.* 8–21.

[3] Zafar Ahmad, Rezaul Chowdhury, Rathish Das, Pramod Ganapathi, Aaron Gregory, and Yimin Zhu. 2022. Brief Announcement: Faster Stencil Computations using Gaussian Approximations. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures.* 291–293.

[4] Zafar Ahmad, Rezaul Chowdhury, Rathish Das, Yushen Huang, and Yimin Zhu. 2023. Fast option pricing using nonlinear stencils. *arXiv preprint arXiv:2303.02317* (2023).

[5] Ghada Alobaidi and Roland Mallier. 2001. Asymptotic analysis of American call options. *International Journal of Mathematics and Mathematical Sciences* 27, 3 (2001), 177–188.

[6] William F. Ames. 2014. *Numerical Methods for Partial Differential Equations*. Academic press.

[7] Rumen Andonov and Sanjay Rajopadhye. 1997. Optimal orthogonal tiling of 2-D iterations. *Journal of Parallel and Distributed computing* 45, 2 (1997), 159–165.

[8] Abdon Atangana and Juan J. Nieto. 2015. Numerical solution for the model of RLC circuit via the fractional derivative without singular kernel. *Advances in Mechanical Engineering* 7, 10 (2015), 1687814015613758.

[9] Joelle Aubin, David F. Fletcher, and Catherine Xuereb. 2004. Modeling turbulent flow in stirred tanks with CFD: the influence of the modeling approach, turbulence model and numerical scheme. *Experimental thermal and fluid science* 28, 5 (2004), 431–445.

[10] Roni Avissar and Roger A Pielke. 1989. A parameterization of heterogeneous land surfaces for atmospheric numerical models and its impact on regional meteorology. *Monthly Weather Review* 117, 10 (1989), 2113–2136.

[11] Louis Bachelier. 1900. Théorie de la spéculation. In *Annales scientifiques de l'École normale supérieure*, Vol. 17. 21–86.

[12] Gurdip Bakshi and Dilip Madan. 2000. Spanning and derivative-security valuation. *Journal of Financial Economics* 55, 2 (2000), 205–238.

[13] Clifford A. Ball and Antonio Roma. 1994. Stochastic volatility option pricing. *Journal of Financial and Quantitative Analysis* 29, 4 (1994), 589–607.

[14] Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula. 2012. Tiling stencil computations to maximize parallelism. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–11.

[15] Timothy J. Barth and Herman Deconinck. 2013. *High-order Methods for Computational Physics*. Vol. 9. Springer Science & Business Media.

[16] Alain Bensoussan. 1984. On the theory of option pricing. *Acta Applicandae Mathematica* 2, 2 (1984), 139–158.

[17] Lorenzo Bergomi. 2015. *Stochastic Volatility Modeling*. CRC Press.

[18] Fischer Black and Myron Scholes. 1973. The pricing of options and corporate liabilities. *Journal of Political Economy* 81, 3 (1973), 637–654.

[19] Uday Bondhugula, Aravind Acharya, and Albert Cohen. 2016. The Pluto+ algorithm: a practical approach for parallelization and locality optimization of affine loop nests. *ACM Transactions on Programming Languages and Systems* 38, 3 (2016), 1–32.

[20] Uday Bondhugula, Vinayaka Bandishti, and Irshad Pananilath. 2017. Diamond tiling: tiling techniques to maximize parallelism for stencil computations. *IEEE Transactions on Parallel and Distributed Systems* 28, 5 (2017), 1285–1298.

[21] Phelim Boyle. 1986. Option valuation using a three-jump process. *International Options Journal* 3 (1986), 7–12.

[22] Phelim P. Boyle. 1977. Options: a Monte Carlo approach. *Journal of Financial Economics* 4, 3 (1977), 323–338.

[23] Terryn Brunelle. 2022. *Parallelizing Tree Traversals for Binomial Option Pricing*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[24] Richard L. Burden, Douglas J. Faires, and Annette M. Burden. 2015. *Numerical Analysis*. Cengage Learning.

[25] Chuang-Chang Chang, San-Lin Chung, and Richard C. Stapleton. 2007. Richardson extrapolation techniques for the pricing of American-style options. *Journal of Futures Markets: Futures, Options, and Other Derivative Products* 27, 8 (2007), 791–817.

[26] Xinfu Chen and John Chadam. 2007. A mathematical analysis of the optimal exercise boundary for American put options. *SIAM Journal on Mathematical Analysis* 38, 5 (2007), 1613–1641.

[27] Zhuliang Chen and Peter A. Forsyth. 2008. A numerical scheme for the impulse control formulation for pricing variable annuities with a guaranteed minimum withdrawal benefit (GMWB). *Numer. Math.* 109, 4 (2008), 535–569.

[28] Rama Cont and Ekaterina Voltchkova. 2005. A finite difference scheme for option pricing in jump diffusion and exponential Lévy models. *SIAM J. Numer. Anal.* 43, 4 (2005), 1596–1626.

[29] The QuantLib contributors. 2022. *QuantLib: a Free/open-Source Library for Quantitative Finance*. https://doi.org/10.5281/zenodo.6461652

[30] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to Algorithms*. MIT Press.

[31] John C. Cox, Stephen A. Ross, and Mark Rubinstein. 1979. Option pricing: a simplified approach. *Journal of Financial Economics* 7, 3 (1979), 229–263.

[32] John Crank and Phyllis Nicolson. 1947. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. In *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 43. Cambridge University Press, 50–67.

[33] Peter A. Cundall and Roger D. Hart. 1992. Numerical modeling of discontinua. *Engineering Computations* (1992).

[34] Michael A. H. Dempster and George S. S. Hong. 2002. Spread option valuation and the fast Fourier transform. In *Mathematical Finance-Bachelier Congress 2000*. Springer, 203–220.

[35] Emanuel Derman and Iraj Kani. 1994. Riding on a smile. *Risk* 7, 2 (1994), 32–39.

[36] Emanuel Derman and Iraj Kani. 1998. Stochastic implied trees: arbitrage pricing with stochastic term and strike structure of volatility. *International Journal of Theoretical and Applied Finance* 1, 01 (1998), 61–110.

[37] Jin-Chuan Duan and Jean-Guy Simonato. 1998. Empirical martingale simulation for asset prices. *Management Science* 44, 9 (1998), 1218–1233.

[38] Darrell Duffie, Jun Pan, and Kenneth Singleton. 2000. Transform analysis and asset pricing for affine jump-diffusions. *Econometrica* 68, 6 (2000), 1343–1376.

[39] Bruno Dupire. 1994. Pricing with a smile. *Risk* 7, 1 (1994), 18–20.

[40] Matteo Frigo and Volker Strumpen. 2005. Cache oblivious stencil computations. In *Proceedings of the 19th International Conference on Supercomputing*. 361–366.

[41] Matteo Frigo and Volker Strumpen. 2009. The cache complexity of multithreaded cache oblivious algorithms. *Theory of Computing Systems* 45, 2 (2009), 203–233.

[42] Dan Galai and Ronald W. Masulis. 1976. The option pricing model and the risk factor of stock. *Journal of Financial Economics* 3, 1-2 (1976), 53–81.

[43] Charles F. Gammie, Jonathan C. McKinney, and Gábor Tóth. 2003. HARM: a numerical scheme for general relativistic magnetohydrodynamics. *The Astrophysical Journal* 589, 1 (2003), 444.

[44] Hélyette Geman, Dilip B. Madan, and Marc Yor. 2001. Time changes for Lévy processes. *Mathematical Finance* 11, 1 (2001), 79–96.

[45] Daozhi Han and Xiaoming Wang. 2015. A second order in time, uniquely solvable, unconditionally stable numerical scheme for Cahn-Hilliard-Navier-Stokes equation. *J. Comput. Phys.* 290 (2015), 139–156.

[46] Steven L. Heston. 1993. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies* 6, 2 (1993), 327–343.

[47] Francis B. Hildebrand. 1987. *Introduction to Numerical Analysis*. Courier Corporation.

[48] Tomoyuki Hirouchi, Tomohiro Takaki, and Yoshihiro Tomita. 2009. Development of numerical scheme for phase field crystal deformation simulation. *Computational Materials Science* 44, 4 (2009), 1192–1197.

[49] Karin Högstedt, Larry Carter, and Jeanne Ferrante. 1999. Selecting tile shape for minimal execution time. In *ACM Symposium on Parallel algorithms and architectures*. 201–211.

[50] Mark H. Holmes. 2007. *Introduction to Numerical Methods in Differential Equations*. Springer.

[51] Peter Hördahl and Frank Packer. 2007. Understanding asset prices: an overview. *Bis Papers* (2007).

[52] John C. Hull. 2003. *Options Futures and Other Derivatives* (5th ed.). Pearson Education India.

[53] Alfredo Ibanez and Fernando Zapatero. 2004. Monte Carlo valuation of American options through computation of the optimal exercise frontier. *Journal of Financial and Quantitative Analysis* 39, 2 (2004), 253–275.

[54] Kenneth R. Jackson, Sebastian Jaimungal, and Vladimir Surkov. 2008. Fourier space time-stepping for option pricing with Lévy models. *Journal of Computational Finance* 12, 2 (2008), 1–29.

[55] Frederick James. 1980. Monte Carlo theory and practice. *Reports on Progress in Physics* 43, 9 (1980), 1145.

[56] Martin T. Johnson. 2010. A numerical scheme to calculate temperature and salinity dependent air-water transfer velocities for any gas. *Ocean Science* 6, 4 (2010), 913–932.

[57] Eugenia Kalnay, Masao Kanamitsu, and Wayman E. Baker. 1990. Global numerical weather prediction at the National Meteorological Center. *Bulletin of the American Meteorological Society* 71, 10 (1990), 1410–1428.

[58] Ioannis Karatzas and Steven E. Shreve. 1998. *Methods of Mathematical Finance*. Vol. 39. Springer.

[59] Serguei S. Komissarov. 2002. Time-dependent, force-free, degenerate electrodynamics. *Monthly Notices of the Royal Astronomical Society* 336, 3 (2002), 759–766.

[60] Steven G. Kou. 2002. A jump-diffusion model for option pricing. *Management science* 48, 8 (2002), 1086–1101.

[61] Sriram Krishnamoorthy, Muthu Baskaran, Uday Bondhugula, Jagannathan Ramanujam, Atanas Rountev, and Ponnuswamy Sadayappan. 2007. Effective automatic parallelization of stencil computations. *ACM SIGPLAN Notices* 42, 6 (2007), 235–244.

[62] Sunil Kumar, Ahmet Yildirim, Yasir Khan, Hossein Jafari, Khosro Sayevand, and Leilei Wei. 2012. Analytical solution of fractional Black-Scholes European option pricing equation by using Laplace transform. *Journal of Fractional Calculus and Applications* 2, 8 (2012), 1–9.

[63] Kenneth Kiprotich Langat, Joseph Ivivi Mwaniki, and George Korir Kiprop. 2019. Pricing options using trinomial lattice method. *Journal of Finance and Economics* 7, 3 (2019), 81–87.

[64] Randall J. LeVeque. 2007. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-state and Time-dependent Problems*. SIAM.

[65] Wen Long, James T. Kirby, and Zhiyu Shao. 2008. A numerical scheme for morphological bed level calculations. *Coastal Engineering* 55, 2 (2008), 167–180.

[66] Francis A. Longstaff and Eduardo S. Schwartz. 2001. Valuing American options by simulation: a simple least-squares approach. *The Review of Financial Studies* 14, 1 (2001), 113–147.

[67] Gabriel J. Lord and Jacques Rougemont. 2004. A numerical scheme for stochastic PDEs with Gevrey regularity. *IMA J. Numer. Anal.* 24, 4 (2004), 587–604.

[68] Roger Lord, Fang Fang, Frank Bervoets, and Cornelis W. Oosterlee. 2008. A fast and accurate FFT-based method for pricing early-exercise options under Lévy processes. *SIAM Journal on Scientific Computing* 30, 4 (2008), 1678–1705.

[69] Henry P. MacKean. 1965. A free boundary problem for the heat equation arising from a problem in mathematical economics. *Industrial Management Review* 6 (1965), 32–39.

[70] Dilip B. Madan, Peter P. Carr, and Eric C. Chang. 1998. The variance gamma process and option pricing. *Review of Finance* 2, 1 (1998), 79–105.

[71] Tareq Malas, Georg Hager, Hatem Ltaief, and David Keyes. 2014. Towards energy efficiency and maximum computational intensity for stencil algorithms using wavefront diamond temporal blocking. *ArXiv* abs/1410.5561 (2014).

[72] Tareq Malas, Georg Hager, Hatem Ltaief, Holger Stengel, Gerhard Wellein, and David Keyes. 2015. Multicore-optimized wavefront diamond blocking for optimizing stencil updates. *SIAM Journal on Scientific Computing* 37, 4 (2015), C439–C464.

[73] André Mangeney, Francesco Califano, Carlo Cavazzoni, and Pavel M. Travnicek. 2002. A numerical scheme for the integration of the Vlasov-Maxwell system of equations. *J. Comput. Phys.* 179, 2 (2002), 495–538.

[74] Kazuhisa Matsuda. 2004. Introduction to Merton jump diffusion model. *Department of Economics, The Graduate Center, The City University of New York, New York* (2004).

[75] Robert C. Merton. 1973. Theory of rational option pricing. *The Bell Journal of Economics and Management Science* (1973), 141–183.

[76] Robert C. Merton. 1976. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics* 3, 1-2 (1976), 125–144.

[77] Nicholas Metropolis and Stanislaw Ulam. 1949. The Monte Carlo method. *J. Amer. Statist. Assoc.* 44, 247 (1949), 335–341.

[78] Ross J. Murray and Ian Simmonds. 1991. A numerical scheme for tracking cyclone centres from digital data. *Australian Meteorological Magazine* 39, 3 (1991), 155–166.

[79] Habib N. Najm, Peter S. Wyckoff, and Omar M. Knio. 1998. A semi-implicit numerical scheme for reacting flow: I. stiff chemistry. *J. Comput. Phys.* 143, 2 (1998), 381–402.

[80] Pedro S. Oliveira. 2014. *The Convolution Method for Pricing American Options under Lévy Processes*. Ph. D. Dissertation. University of Lisbon.

[81] Ekanathan Palamadai Natarajan, Maryam Mehri Dehnavi, and Charles Leiserson. 2017. Autotuning divide-and-conquer stencil computations. *Concurrency and Computation: Practice and Experience* 29, 17 (2017), e4127.

[82] Tao Pang. 1999. *An Introduction to Computational Physics*. American Association of Physics Teachers.

[83] Laetitia Paoli and Michelle Schatzman. 2002. A numerical scheme for impact problems I: The one-dimensional case. *SIAM J. Numer. Anal.* 40, 2 (2002), 702–733.

[84] perf [n. d.]. perf: Linux profiling with performance counters. https://perf.wiki.kernel.org/index.php/Main_Page.

[85] Gabriel Peyré. 2011. The numerical tours of signal processing-advanced computational signal and image processing. *IEEE Computing in Science and Engineering* 13, 4 (2011), 94–97.

[86] Din Prathumwan and Kamonchat Trachoo. 2020. On the solution of two-dimensional fractional Black-Scholes equation for European put option. *Advances in Difference Equations* 2020, 1 (2020), 1–9.

[87] Michel Rappaz, Michel Bellet, and Michel Deville. 2010. *Numerical Modeling in Materials Science and Engineering*. Vol. 32. Springer Science & Business Media.

[88] Richard J. Rendleman. 1979. Two-state option pricing. *The Journal of Finance* 34, 5 (1979), 1093–1110.

[89] Ludovic Renson, Gaëtan Kerschen, and Bruno Cochelin. 2016. Numerical computation of nonlinear normal modes in mechanical engineering. *Journal of Sound and Vibration* 364 (2016), 177–206.

[90] Lewis F. Richardson. 1911. The approximate arithmetical solution by finite differences with an application to stresses in masonry dams. *Philosophical Transactions of the Royal Society of America* 210 (1911), 307–357.

[91] Lewis F. Richardson and John A. Gaunt. 1927. VIII. The deferred approach to the limit. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* 226, 636-646 (1927), 299–361.

[92] André Robert. 1981. A stable numerical integration scheme for the primitive meteorological equations. *Atmosphere-Ocean* 19, 1 (1981), 35–46.

[93] André Robert. 1982. A semi-Lagrangian and semi-implicit numerical integration scheme for the primitive meteorological equations. *Journal of the Meteorological Society of Japan. Ser. II* 60, 1 (1982), 319–325.

[94] Wolfgang J. Runggaldier. 2003. Jump-diffusion models. In *Handbook of Heavy Tailed Distributions in Finance*. Elsevier, 169–209.

[95] Paul A. Samuelson, Alison Etheridge, Mark Davis, and Louis Bachelier. 2011. *Louis Bachelier's Theory of Speculation: The Origins of Modern Finance*. Princeton University Press.

[96] Katsuto Sato, Hiroyuki Takizawa, Kazuhiko Komatsu, and Hiroaki Kobayashi. 2010. Automatic tuning of CUDA execution parameters for stencil processing. *Software Automatic Tuning: From Concepts to State-of-the-Art Results* (2010), 209–228.

[97] Granville Sewell. 2005. *The Numerical Solution of Ordinary and Partial Differential Equations*. Vol. 75. John Wiley & Sons.

[98] William F. Sharpe. 1978. *Investments*. Prentice Hall.

[99] Steven Shreve. 2005. *Stochastic calculus for finance I: the binomial asset pricing model*. Springer Science & Business Media.

[100] Steven E. Shreve. 2004. *Stochastic Calculus for Finance II: Continuous-time Models*. Vol. 11. Springer.

[101] Clifford W. Smith. 1976. Option pricing: a review. *Journal of Financial Economics* 3, 1-2 (1976), 3–51.

[102] Dale Snider and Sibashis Banerjee. 2010. Heterogeneous gas chemistry in the CPFD Eulerian-Lagrangian numerical scheme (ozone decomposition). *Powder Technology* 199, 1 (2010), 100–106.

[103] Elias M. Stein and Jeremy C. Stein. 1991. Stock price distributions with stochastic volatility: an analytic approach. *The Review of financial studies* 4, 4 (1991), 727–752.

[104] John C. Strikwerda. 2004. *Finite Difference Schemes and Partial Differential Equations*. SIAM.

[105] Rudolph Szilard. 2004. Theories and applications of plate analysis: classical, numerical and engineering methods. *Appl. Mech. Rev.* 57, 6 (2004), B32–B33.

[106] Allen Taflove and Susan C. Hagness. 2005. *Computational Electrodynamics: the Finite-difference Time-domain Method*. Artech house.

[107] Yuan Tang, Rezaul Chowdhury, Bradley Kuszmaul, Chi-Keung Luk, and Charles Leiserson. 2011. The Pochoir stencil compiler. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures*. 117–128.

[108] Yuan Tang, Rezaul Chowdhury, Chi-Keung Luk, and Charles Leiserson. 2011. Coding stencil computations using the Pochoir stencil-specification language. In *USENIX Workshop on Hot Topics in Parallelism*.

[109] Jos Thijssen. 2007. *Computational Physics*. Cambridge University Press.

[110] James W. Thomas. 2013. *Numerical Partial Differential Equations: Finite Difference Methods*. Vol. 22. Springer Science & Business Media.

[111] John N. Tsitsiklis and Benjamin Van R. 2001. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks* 12, 4 (2001), 694–703.

[112] Pierre Van M. 1976. On optimal stopping and free boundary problems. *Archive for Rational Mechanics and Analysis* 60, 2 (1976), 101–148.

[113] Ursula Van R. 2012. *Numerical Methods in Computational Electrodynamics: Linear Systems in Practical Applications*. Vol. 12. Springer Science & Business Media.

[114] Luminita A. Vese and Stanley J. Osher. 2002. Numerical methods for p-harmonic flows and applications to image processing. *SIAM J. Numer. Anal.* 40, 6 (2002), 2085–2104.

[115] Franz J. Vesely. 1994. *Computational Physics*. Springer.

[116] D. Vijayaraghavan and Theo Keith. 1990. An efficient, robust, and time accurate numerical scheme applied to a cavitation algorithm. (1990).

[117] Vincent M. Weaver, Dan Terpstra, Heike McCraw, Matt Johnson, Kiran Kasichayanula, James Ralph, John Nelson, Phil Mucci, Tushar Mohan, and Shirley Moore. 2013. PAPI 5: Measuring power, energy, and the cloud. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE. https://doi.org/10.1109/ispass.2013.6557155

[118] Joachim Weickert. 2001. Applications of nonlinear diffusion in image processing and computer vision. *Acta Math. Univ. Comenianae* 70, 1 (2001), 33–50.

[119] Michael E. Wolf and Monica S. Lam. 1991. A data locality optimizing algorithm. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. 30–44.

[120] Michael E. Wolf, Dror E. Maydan, and Ding-Kai Chen. 1996. Combining loop transformations considering caches and scheduling. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 274–286.

[121] Michael J. Wolfe. 1987. Iteration space tiling for memory hierarchies. *Parallel Processing for Scientific Computing* 357 (1987), 361.

[122] David Wonnacott. 2002. Achieving scalable locality with time skewing. *International Journal of Parallel Programming* 30, 3 (2002), 181–221.

[123] Lixia Yuan and Om P. Agrawal. 2002. A numerical scheme for dynamic systems containing fractional derivatives. *Journal of Vibration and Acoustics* 124, 2 (2002), 321–324.

[124] Jianfeng Zhang et al. 2004. A numerical scheme for BSDEs. *Annals of Applied Probability* 14, 1 (2004), 459–488.

[125] Jiyuan Zhang, Tze Meng Low, Qi Guo, and Franz Franchetti. 2015. A 3 d-stacked memory manycore stencil accelerator system. (2015).

[126] Peng Zhang, Sze Chun Wong, and Chi-Wang Shu. 2006. A weighted essentially non-oscillatory numerical scheme for a multi-class traffic flow model on an inhomogeneous highway. *J. Comput. Phys.* 212, 2 (2006), 739–756.

[127] Song-Ping Zhu. 2006. A new analytical approximation formula for the optimal exercise boundary of American put options. *International Journal of Theoretical and Applied Finance* 9, 07 (2006), 1141–1177.

[128] Song-Ping Zhu and Zhi-Wei He. 2007. Calculating the early exercise boundary of American put options with an approximation formula. *International Journal of Theoretical and Applied Finance* 10, 07 (2007), 1203–1227.

[129] Oleksandr Zhylyevskyy. 2010. A fast Fourier transform technique for pricing American options under stochastic volatility. *Review of Derivatives Research* 13, 1 (2010), 1–24.

[130] Mohammad Zubair and Ravi Mukkamala. 2008. High performance implementation of binomial option pricing. In *Computational Science and Its Applications-ICCSA 2008: International Conference, Perugia, Italy, June 30-July 3, 2008, Proceedings, Part I 8*. Springer, 852–866.

A Artifact description

A.1 Requirement

The experiments in this paper were carried out on Stampede2 [1] SKX nodes. Please follow the instructions below to set up your own environment. Note that if the experiments are performed on a machine with a system specification different from the one used for our experiments in this paper, the numbers may vary, but the trends should remain similar.

Getting the Artifact: The artifact is available on Zenodo: https://zenodo.org/records/10531146

Benchmark Software

- BOPM: We compared our BOPM implementation with the Par-bin-ops library. Note that this library requires 'gcc' version 9+ for compilation. Manual download and installation of 'googletest' and 'parlaylib' are needed in the 'external' directory.
- BSM: Vanilla implementation is available inside the corresponding source code.
- TOPM: A vanilla implementation of TOPM is inside `amer-call-TOPM-Vanilla.cpp` and the input parameters are the same as the original `amer-call-TOPM.cpp`.

Software Requirement

- General dependency: Intel C++ compiler, Intel MKL, OpenMP
- Cache miss measurement: PAPI Library
- Energy measurement: Perf Tool

Note: Installation of PAPI and Perf requires sudo access.

Source Package Files

- `amer-call-BOPM.cpp`: American call option pricing under the binomial option pricing model
- `amer-call-TOPM.cpp`: American call option pricing under the trinomial option pricing model
- `amer-put-BSM.cpp`: American put option pricing under the Black-Scholes-Merton option pricing model
- `run_experiments.sh`: Script to reproduce the results in the paper

Note: For each of the three files above [`X.cpp`] we have a corresponding [`X-papi.cpp`] file used to count the number of cache misses. We also provide a helper file `papilib.h` for cache miss computation.

A.2 Execute Scripts

Runtime and Cache Misses

- BOPM and TOPM:

```
./test [T] [Tbase] [numThreads] [verify]
```

- test is the executable file name
- [T] - an integer (> 0), number of steps
- [Tbase] - an integer (≥ 4), is the base-case size where recursion stops and iterative kernels are executed
- [numThreads] - an integer (> 0), number of concurrent threads
- [verify] - Boolean (0/1), if 1 is provided, the result from the iterative implementation is computed to compare with our recursive implementation

Example:

```
./test 40 4 1 0
```

- BSM:

```
./test [T] [Tbase] [w] [numThreads] [verify]
```

- [w] - a positive float; it is a non-dimensional scalar calculated from risk-free rate and volatility; see Section 4.2 for details

Example:

```
./test 32 4 0.5 1 0
```

Setting Environment Variables

To set the number of concurrent threads:

```
export OMP_NUM_THREADS = [numThreads]
set MKL_NUM_THREADS = [numThreads]
```

Check Energy Consumption

```
perf stat -a -e cycles,instructions,cache-
misses,power/energy-cores/,msr/pperf/,power/
energy-ram/,power/energy-pkg/ ./test [T] [
Tbase] [numThreads] [verify]
```

Example:

```
perf stat -a -e cycles,instructions,cache-
misses,power/energy-cores/,msr/pperf/,power/
energy-ram/,power/energy-pkg/ ./test 10000 16
1 0
```

Reproduce the Results in the Paper

```
sh run_experiments.sh
```

The results will be saved under the `./results` folder in their respective `.csv` files.

Interpreting the Output

```
stencil: 0.50072 0.49928
rec result: 6.85587
T = 10000, T_BASE = 4, numThreads = 1, time =
0.009
```

On the first line, we print the stencil coefficient values. On the second line, we print the result computed by our algorithm. On the third line, we print the running time (in seconds) along with the input parameters.

A.3 Checking Results

We provide three types of experimental results: parallel running time (Figure 5 in Section 5), energy consumption (Figure 6 in Section 5), and cache misses (Figure 7 in Section 5). For each value of T, we report the best values after experimenting with varying the number of threads and different base case sizes. For simplicity, you can use the base case size of

16, and set the number of threads to 1 for serial execution and 16 for parallel execution. The reported values of T in the plots are from 2^{10} to 2^{19} (i.e., $1024, 2048, 4096, 8192, \ldots$).

To compare the values against the benchmark:

- For BSM, vanilla implementations can be used by setting the [verify] parameter to 1
- For TOPM, the vanilla implementation is available in `amer-call-TOPM-Vanilla.cpp`
- For BOPM, the Par-bin-ops library needs to be executed