Parallel Black-Scholes Solver on Adaptive Sparse Grids

Sneha Jaiswal, Anushka Parmanand
https://418-cmu-26.github.io/

Summary:

We will implement a parallel algorithm to solve the Black-Scholes equation for options pricing using the OpenMP programming model, and potentially extending to CUDA. We will first focus on parallelizing over a sparse grid with finite elements, which has a hierarchical tree structure and thus irregular memory access patterns; if time permits, we may also implement algorithms to account for non-constant (i.e. Heston) volatility.

Background:

A financial option gives the holder the right to buy or sell a stock at a fixed price K (the *strike price*) at some future date. The question we want to answer computationally is: what is this contract worth today? The Black-Scholes equation is a PDE whose solution $u(S, t)$ gives the option price as a function of the current stock price S and time t.

For a single stock, the Black-Scholes PDE has a known closed-form solution, so there is nothing to compute. The problem becomes interesting when the option depends on $d \geq 2$ stocks simultaneously (a *basket option* — think an option on a portfolio of assets). In that case the PDE has d spatial dimensions, one per stock price, and we must solve it numerically on a grid. The trouble is that a regular Cartesian grid with N points per dimension needs N^d total points which is computationally infeasible.

A sparse grid deals with this by being selective about which grid points to include. The Black-Scholes solution is smooth in most of the domain, so we only need fine resolution where the solution is changing rapidly. A sparse grid uses a hierarchical basis: it starts with a very coarse grid covering the whole domain, then adds finer grid points only where they are needed. With this approach the total number of points drops from $O(N^d)$ to roughly $O(N \log^{d-1} N)$, while losing minimal accuracy.

The main computation on a sparse grid is called the Up-Down algorithm. It's a two-pass sweep over the grid: in the *Up* pass, information flows from fine grid levels to coarse ones; in the *Down* pass, it flows back. The pseudocode is:

```
for each dimension k do
    for l = l_max downto 1 do                          ▷ Up pass: fine to coarse
        for each grid node at level l do               ▷ All nodes at the same level are independent
            update parent node at level l − 1
        end for
    end for
    for l = 1 to l_max do                              ▷ Down pass: coarse to fine
        for each grid node at level l do               ▷ All nodes at the same level are independent
            update child nodes at level l + 1
        end for
    end for
end for
```

The key structure for parallelism is that all nodes at the same level are independent and can be processed simultaneously, but level l cannot begin until level $l+1$ is fully done (Up pass) or level $l-1$ is done (Down pass). This means we can parallelize within a level, but need a synchronization barrier between levels.

The Black-Scholes payoff function (e.g., $\max(S-K,0)$ for a call option) has a sharp kink at the strike price K where the derivative is discontinuous. Uniform sparse grids converge slowly near this kink, so we use adaptive refinement. We add extra grid points near the kink and use fewer elsewhere. This makes the grid more accurate but also more irregular making the parallel problem more interesting.

Challenges:

In practice, the sparse grid Up-Down algorithm is difficult to parallelize efficiently:

- **Random memory access:** In a sparse grid, each node's parents are at non-contiguous memory locations that depend on the node's level and index. So we get frequent cache misses and bad memory bandwidth. Figuring out how to lay out the grid data to reduce this is a key part of the project.
- **Workload imbalance:** In an adaptive grid, some regions (near the payoff kink) are refined much more deeply than others. If we split work by level, threads will have a large variance in the amount of work they have and lots of idle time at barriers. Another challenge is work assignment and scheduling to overcome this issue.
- **Synchronization barriers:** Between every pair of adjacent levels, all threads must fully finish before any thread can move on. When levels are small (few nodes), the time spent waiting at barriers can dominate the time doing actual work.
- **Low arithmetic intensity:** Each node in the Up-Down sweep does few additions and multiplications, but requires loading values from several non-adjacent memory locations. The ratio of arithmetic operations to memory accesses is low, which means parallelization is bottlenecked by memory bandwidth rather than compute. This is the opposite of what GPUs and many-core systems are optimized for, and it limits how much speedup we can realistically get just by adding more cores.
- **Stretch goal: Stochastic volatility makes the grid coupling messier.** The basic Black-Scholes model assumes volatility is constant. The Heston model instead lets volatility vary randomly, adding a second dimension. This introduces a coupling term between the stock-price and volatility dimensions that breaks the clean dimension-by-dimension structure of the Up-Down algorithm, requiring each node to interact with neighbors in both directions at once.
- **Stretch goal: American options change which grid points are active each step.** A European option can only be exercised at expiry. An American option can be exercised at any time, which means at each timestep we have to figure out which grid points are in the "exercise now" region and which are in the "hold" region. This boundary moves around every iteration and is not known in advance, so the set of active work items is different each time, adding data-dependent irregularity.

Resources:

In terms of hardware, we will utilize the GHC machines and potentially the PSC machines for our OpenMP experiments (in C++). If we reach out stretch goal of expanding to CUDA, we will use NVIDIA RTX 2080 GPUs.

Algorithm and implementation references:

1. Heinecke, Schraufstetter, and Bungartz. "A highly parallel Black-Scholes solver based on adaptive sparse grids." *Int. J. Computer Mathematics*, 2012. (link) primary reference for the algorithm and a performance benchmark to compare against.
2. Pflüger. "Spatially Adaptive Sparse Grids for High-Dimensional Problems." PhD thesis, TU Munich, 2010. (free PDF) textbook for sparse grid data structures and the Up-Down algorithm.
3. Schraufstetter and Bungartz. "Parallelizing a Black-Scholes solver based on finite elements and sparse grids." *PPAM 2009*. (link) an earlier OpenMP version of similar solver (can remove if u want)
4. In 't Hout and Foulon. "ADI finite difference schemes for option pricing in the Heston model with correlation." arXiv:0811.3427. (free PDF) for the Heston PDE for stochastic volatility.
5. The **SG++** library (sgpp.sparsegrids.org) CPU sparse grid library.

Goals and Deliverables:

We PLAN TO ACHIEVE the following:

- A correct serial C++ implementation of the sparse grid Black-Scholes solver for basket options on $d = 2$ and $d = 3$ stocks, verified using the SG++ library and known closed-form prices.
- A parallel version of the Up-Down algorithm in the OpenMP shared address space model, using a level-by-level approach with both static and dynamic task scheduling, to directly compare their performance, and assess computation speedup from increasing processors.
- Adaptive refinement near the payoff kink, and a measurement of how much this irregularity hurts parallel efficiency compared to the uniform sparse grid.
- Performance analysis: speedup plots for $d = 2, 3, 4$ at various grid resolutions on 1, 2, 4, and 8 cores; analysis showing memory-bandwidth bottleneck; and a comparison against a plain uniform finite difference grid to demonstrate utility of sparse grids.

We HOPE TO ACHIEVE:

- Heston stochastic volatility: Modify solver so that volatility is a random variable (the Heston model).
- American option pricing: Allow early exercise, which means solving an extra constraint at each timestep to figure out where the early-exercise boundary is.
- GPU extension: Implement the parallel Black-Scholes solver in CUDA to compare absolute performance and speedup to CPU experiments. We expect that the irregular memory access pattern will deteriorate GPU performance on this task.

Performance target: Heinecke et al. (2012) achieve near-linear speedup on 16 cores with OpenMP on uniform sparse grids. We aim for at least 7× speedup on 8 cores for the uniform case; once our parallel implementation achieves near-linearity, this project will focus on uncovering the extent to which adaptive refinement, which improves pricing accuracy, degrades performance, as well as investigating how well dynamic scheduling can recover speedup. We do not set numerical performance targets for the Heston and American stretch goals, as we just hope to discover if there is overhead relative to the simpler European case.

Platform Choice:

We are focusing on the OpenMP model on multi-core CPU for two reasons. First, the level-by-level Up-Down sweep maps directly onto OpenMP parallel for-loops with barriers, so the code structure stays readable while we work out the algorithm. In particular, the shared address space model would outperform a message passing model because it avoids intense data replication across local memory in each thread and the heavy cost of broadcasting or sharing updates at the barrier at each level. Second, CPU profiling tools like perf and VTune make it easy to see exactly how much execution time goes towards cache misses, barrier waits, idle threads, etc. This helps us understand and explain the bottlenecks before adding GPU complexity.

If we reach the GPU extension, it is interesting precisely because the algorithm is a bad fit for GPUs: the irregular memory accesses prevent coalescing, and the barrier-heavy level structure means threads cannot stay busy independently. Demonstrating and quantifying that mismatch on an RTX 2080 would highlight the potential performance gains on multi-core CPUs for this problem.

Schedule:

Week	**Tasks**
Week 1(Mar 25 – Apr 1)	Read the Pfluger thesis (Ch. 2–4) and Heinecke et al. to understand the data structure and algorithm. Build the sparse grid in C++: store nodes by level and index, implement neighbor lookups, handle boundaries. Test on a simple 1D case against SG++.
Week 2(Apr 2 – Apr 8)	Implement the serial Up-Down algorithm and time-stepping. Check 2D and 3D option prices match known values. Profile the serial code to find the main bottleneck (prob memory access).
Week 3(Apr 9 – Apr 15)	**Milestone.** First OpenMP parallel version, with static scheduling by level. Do experiments
Week 4(Apr 16 – Apr 22)	Add adaptive refinement near the strike price. Switch to dynamic scheduling and measure the difference. Run experiments/profile more
Week 5(Apr 23 – Apr 29)	Stretch goals: add Heston stochastic volatility then American option early exercise. Start CUDA if got time.
Week 6(Apr 30 – May 5)	Finish all experiments. Write up results and make the poster. Produce plots, adaptive vs. uniform comparisons, etc.