TITLE: Parallel Longstaff-Schwartz American Option Pricing on CPU & GPU

URL:https://github.com/Yunfan-Wang/CMU-15418-Project-Longstaff-Schwartz-Solver

SUMMARY:

We will implement and optimize the Longstaff-Schwartz Monte Carlo (LSM) algorithm for pricing american style financial options on parallel architectures. The goal is to evaluate and compare two optimized implementations on multicore CPUs using MPI and NVIDIA GPUs (CUDA). We will study how different data layouts, reduction strategies and degrees of parallelism would impact the performance.

BACKGROUND:

The american options allow holders to exercise at any time before maturity. As compared to European options, which has a fixed date of exercise, American options are significantly harder to price. The Longstaff-Schwartz Method prices these options by combining massively parallel Monte Carlo path simulation, backward dynamic programming and regression-based continuation value estimation together. The steps:

1. Simulate N independent asset price paths using Geometric Brownian Motion:
$$S_{t+\Delta t} = S_t \exp\left((r - \tfrac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t} Z \right)$$
2. Backward induction over time steps:
   Solve the regression by applying backward regression: β = (X^T X)^(-1) X^TY
   Compare continuation value C = βθ, $(where\ \theta is\ the\ basis\ function)$ with immediate exercise value.
3. Compute the final price as discounted average exercise payoff[1]

The parallelism lies in

1. Simulation: since each path in Monte Carlo path simulation is independent.
2. Regression building. The X^T X and X^T Y are sums over all paths, which is a reduction problem.
3. Exercise decision is perpath, which is fully parallel.

THE CHALLENGE:

- Describe the workload:

The original algorithm is highly sequential because it goes backwards through time and every value at the current time step depends on all of the values from the previous iteration.

However, there are places where parallelism can be exploited:

1. Regression: At each time step t, as is described above, the backward regression is represented as matrix multiplications. Each path contributes to the same small matrix/vector. Threads can compute local partial sums in parallel. The kind of parallelism is similar to what assignment2 did.
2. Monte carlo algorithm is inherently a explicit chance of parallelism but the branching decision to make it efficient can be somewhat challenging.
3. There are more parallelism to be exploit in the improved implementation.

A challenge for the MPI implementation is how to overlap computation with communication because a lot of the computation depends on the messages that we're going to receive.
A challenge for the CUDA implementation is that the algorithm only considers options that are "in the money", which may lead to divergent execution because we can stop considering a path part way through the algorithm.

- Describe constraints:

A challenge that comes from the MPI system is load balancing. The algorithm stops considering some paths part way through the algorithm, meaning the load will change over time.

A challenge for the CUDA implementation is that the blocks must execute independently, so we must be careful to identify dependencies and make sure that there are no dependencies across blocks.

RESOURCES:

Original long staff schwartz paper:
https://people.math.ethz.ch/~hjfurrer/teaching/LongstaffSchwartzAmericanOptionsLeastSquareMonteCarlo.pdf

We can use this as starter code, but make our own implementation:
https://github.com/NVIDIA-developer-blog/code-samples/tree/master/posts/american-options

Article about GPU accelerated algorithm:
https://faculty.fordham.edu/rchen/gpu.pdf

Papers about parallelizing algorithm:
https://www.researchgate.net/publication/220950873_Optimizations_in_financial_engineering_The_Least-Squares_Monte_Carlo_method_of_Longstaff_and_Schwartz

https://link.springer.com/article/10.1007/s11227-015-1451-7

Paper about modified algorithm that's more amenable to parallelization:
https://arxiv.org/pdf/1404.1180

GOALS AND DELIVERABLES:

Plan to achieve:

- Sequential implementation (25%)
- CUDA implementation (50%)
- MPI implementation (75%)
- Comparisons and analysis between implementations (100%)

Hope to achieve:

- I hope to have our implementations outperform the implementation from NVIDIA on github because I want to do better than code thats publicly available (125%)
- Implement a modified algorithm from a paper and compare it to the implementations of the original algorithm. This will allow us to show how changing the algorithm might lead to better performance. (150%)

Questions to answer:

- Whether MPI or CUDA is better suited for this algorithm
  - Which is faster
  - Which gives better results
  - Which is more power efficient

PLATFORM CHOICE:

We have chosen to use CUDA and MPI.
CUDA is good for our needs because there is a lot of opportunities for SIMD execution in the algorithm in the linear regression and vector and matrix operations.

MPI is also good because although the algorithm is highly sequential, we can partition the work among processors, but careful communication is required between the processors to maintain good results.

SCHEDULE:

Week 1 (Nov 17 - Nov 23)	• Sequential implementation • Start CUDA implementation

Week 2 (Nov 24 - Nov 30)	● Finish CUDA implementation ● MPI implementation
Week 3 (Dec 1 - Dec 7)	● Profile and performance debug MPI implementation ● Run experiments and write report
Due date: Dec 8	● Implement modified algorithm if we have time

DIVISION OF WORK:

We will together find a exemplar sequential implementation as the baseline and proper referencing it.
Then Ronnie will finish mpi part and Yunfan will finish the cuda part.
Afterwards, we will compare the performance of each others implementation as well as with sequential implementation to together finish the report.

Reference:

[1] Herrera, C. & Paulot, L. (2014). *Parallel American Monte Carlo*. Preprint available at arXiv:1404.1180.