COSC7502 Major Project

HPC in Monte Carlo Simulation for European Basket Option Pricing

Authored by Joel Thomas

Introduction to Options Pricing

- A type of formal financial agreement
- Have two interested parties desiring to facilitate a potential transaction on an underlying asset S
- Transaction must be at some fixed predefined price K
- Must occur prior to or on the contract expiration date T
  - American vs. European options contracts
- Option price at maturity is C_T, price today is C_0

European Basket Option

- Contract payoff at maturity for standard European call option:
  - $C_T = \max\{S_T - K, 0\} = (S_T - K)^+$
- Contract payoff at maturity for standard European basket call option:
  - $C_T = \left(\frac{1}{D}\sum_{d=1}^{D} S_T^{(d)} - K\right)^+$
  - D potentially correlated (or uncorrelated) assets rather than just a single asset

Challenges

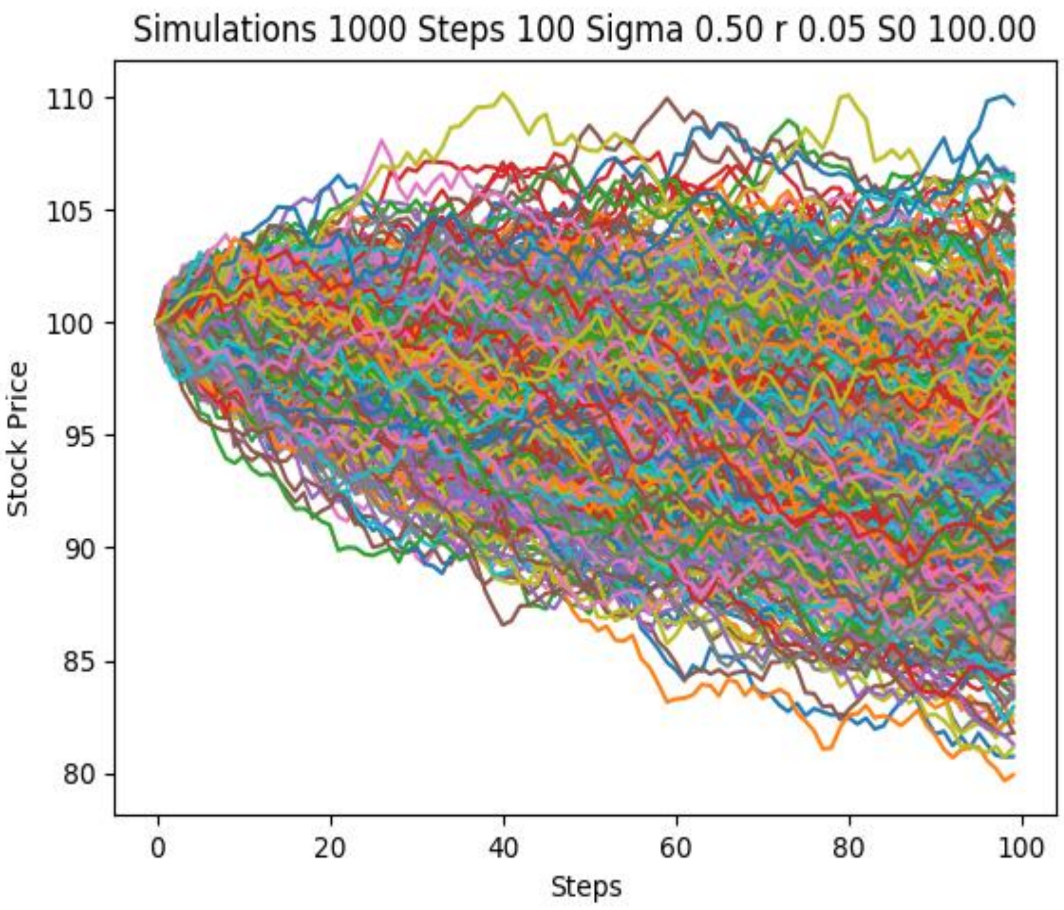
- Want to find reasonable approximation for C_0
- Essentially need to find a way to calculate time T prices for not just one but each of the D assets
  - Prices on financial assets are typically stochastic processes
  - Must respect any correlation dependencies between the assets
  - But how to calculate future price of something completely random?
- Potential solution – use Monte Carlo simulation to simulate multiple potential price paths up to time T for each of the D assets

MC Simulation combined with Euler Timestepping

- Revisit background mathematics explained extensively in report
- Assume each asset is a Geometric Brownian Motion with a common SDE
  - SDE given by $dS_t^{(d)} = \mu S_t^{(d)} dt + \sigma S_t^{(d)} dW_t^{(d)}$
  - Note shared μ, σ
- Corresponding Euler scheme used:
  - $\hat{S}_{n+1}{}^{(d,m)} = \hat{S}_n{}^{(d,m)} + \mu \hat{S}_n{}^{(d,m)} \Delta t + \sigma \hat{S}_n{}^{(d,m)} (LZ_n)^{(d,m)} \sqrt{\Delta t}$
  - For $1 \leq d \leq D,\ 0 \leq n \leq N-1, 1 \leq m \leq M$
  - $D =$ number of assets, $N =$ number of simulation timesteps, $M =$ number of MC samples
- Respects any underlying correlation dependencies between assets via Cholesky factorisation of correlation matrix P to obtain L
  - LZ_n stores correlated standard normal random numbers satisfying $Corr(LZ_n) = LL^T = \mathrm{P}$

Simulating Single Asset Example

- $D = 1$
- $N = 100$
- $M = 1000$



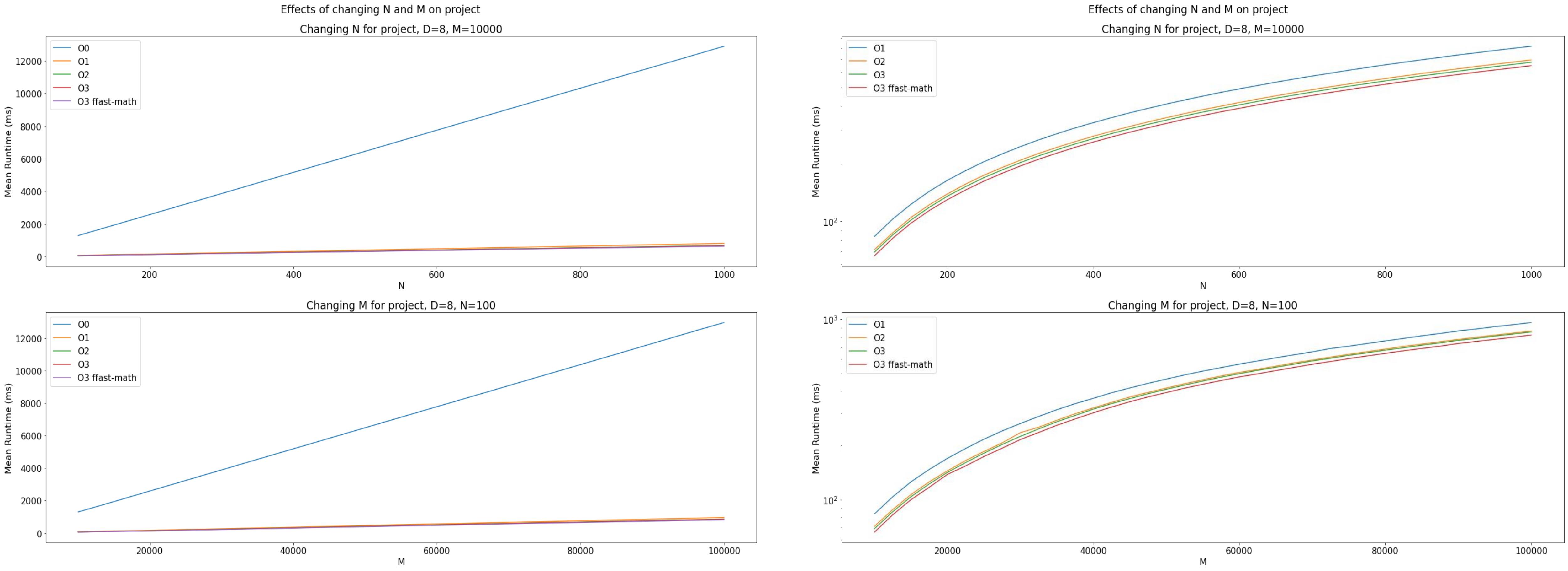
Serial Implementation Details

- Pseudocode provided in Table 1 of report
- Implemented in C++
- Use of high-performance Eigen C++ library for basic linear algebra operations
  - Vector/matrix arithmetic, row-wise and column-wise operations, Cholesky factorisation of P, etc.
- Use of Ziggurat algorithm for faster standard normal RNG
  - Used to generate each Z_n faster
- Use of $gprof$ profiler showed methods from both these libraries at towards top of the flat profile.

Serial Implementation Details

- Manually implemented optimisations:
  - Advanced (faster) matrix initialisations at each time sub-interval t_n using a subroutine from the Eigen library
  - Avoided recalculating constants appearing within loops by moving them outside
  - Removed unnecessary type conversions from $double$ to $float$

Serial Implementation Performance results



Parallel Implementation Details

- Approach taken was to parallelise the serial implementation fully as is
- Developed using NVIDIA's CUDA parallel programming platform in CUDA C++
- Pseudocode provided in Table 2 of report
- Used CUDA-X libraries including $cuRAND$, $cuSOLVER$, $cuBLAS$ and $thrust$
    - For standard normal RNG, performing Cholesky decomposition, matrix/vector arithmetic and parallel reduction algorithms respectively

Verification Procedure

- Using special case of $\mathrm{P} = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$, shared $S_0 = 100, K = 100, \mu = r = 0.02, T = 1, \sigma = 0.3$ for all $D = 8$ assets

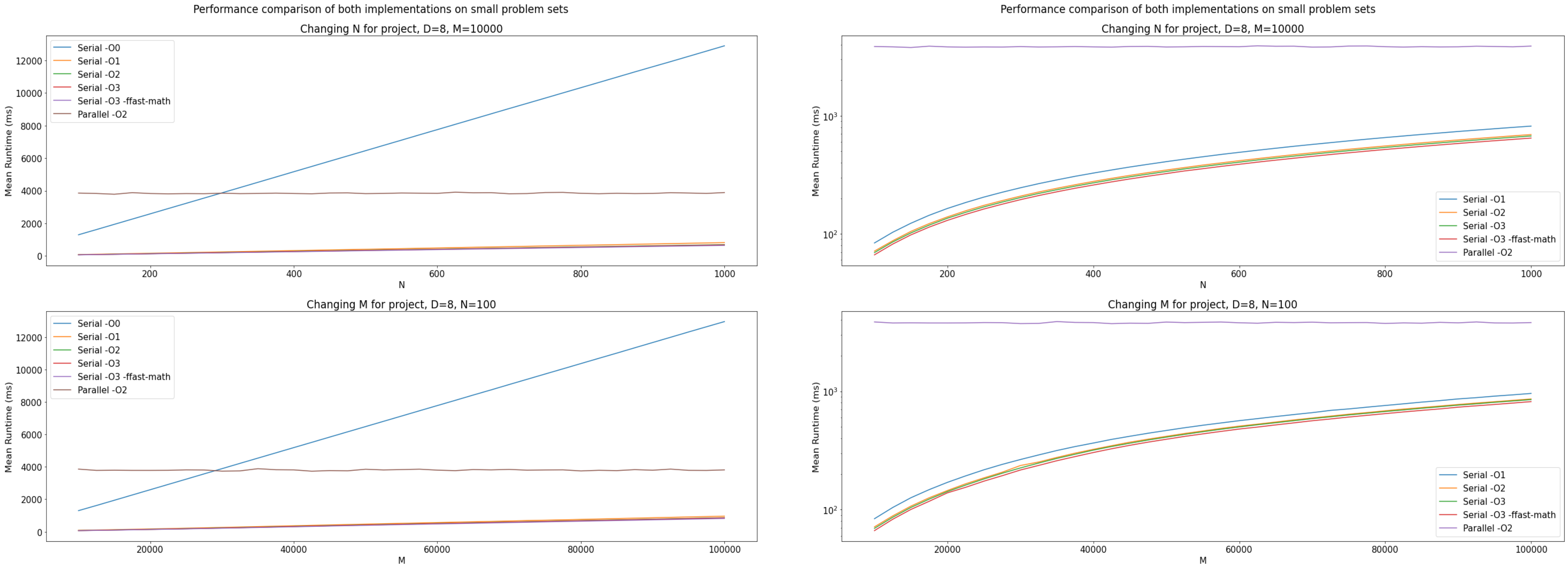
Experimentation Plan

- So far, all of the parallel implementation’s construction, compiling, execution and debugging was done on a personal Windows PC using an NVIDIA RTX 3060 Ti GPU
- Aim to test parallel implementation much more thoroughly using one of the many NVIDIA V100s available on SMP’s Getafix cluster
- First step was conducting special case verification procedure and comparing both implementations’ output across a range of random N, M values to ensure output correctness

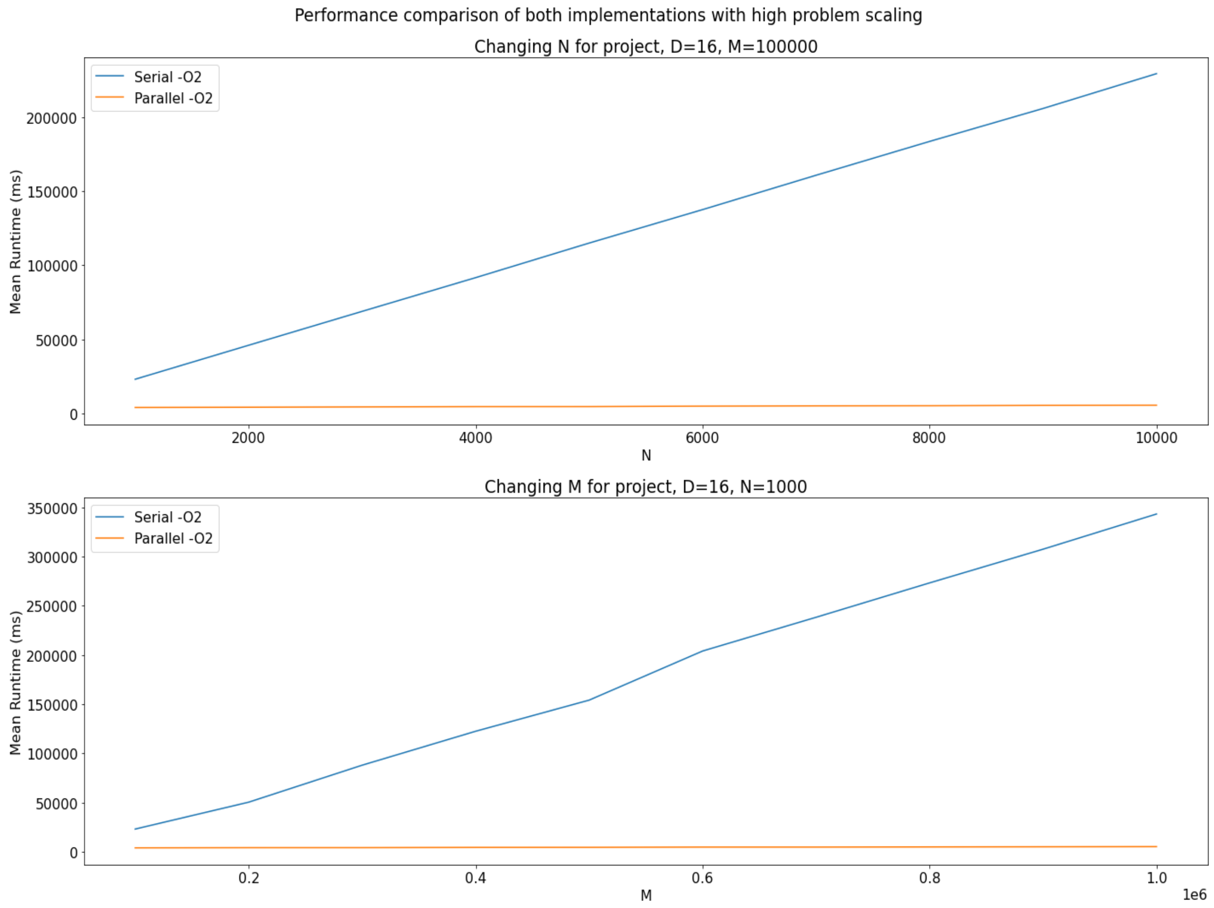
Experimentation Plan

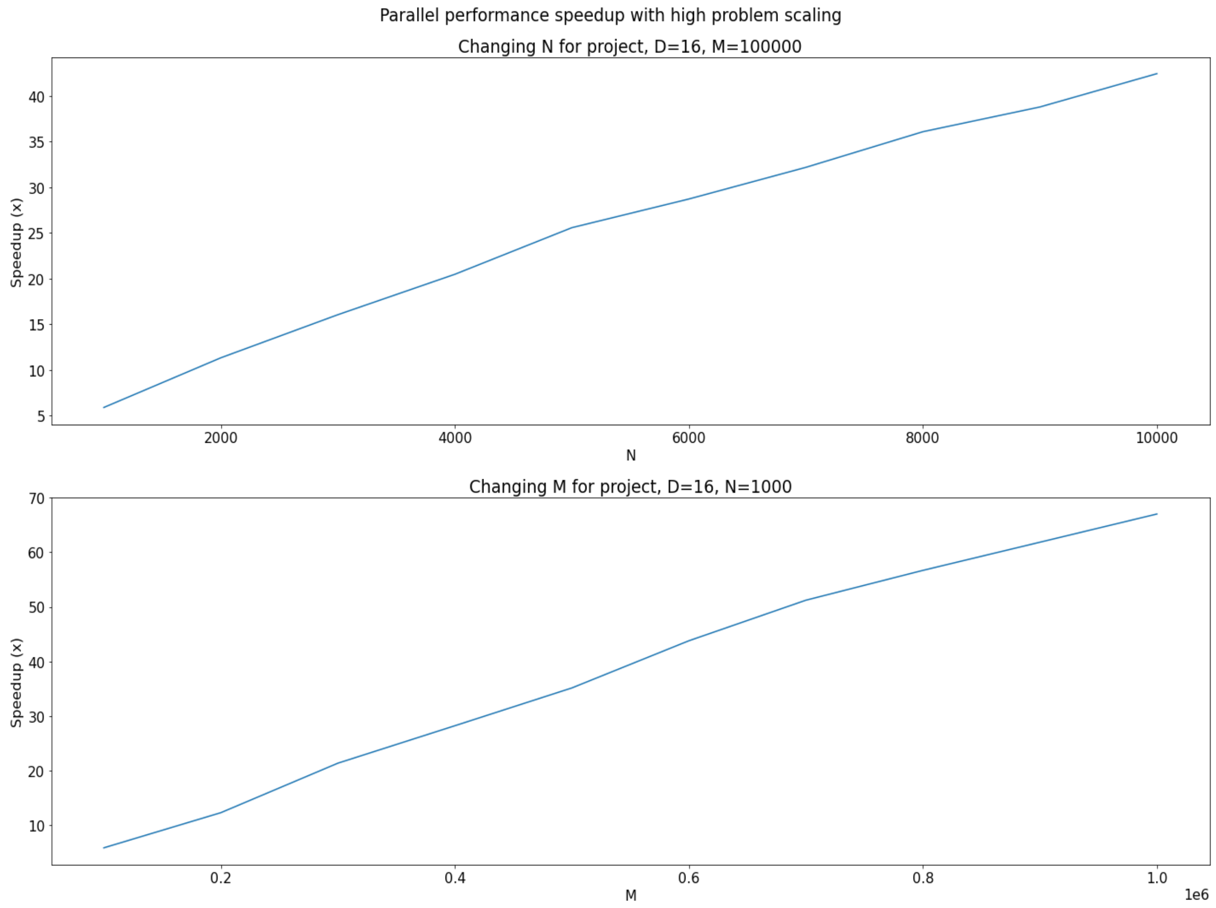
- Full experimentation plan arrived at:
  - Test the parallel implementation on the same set of D, N, M values that were used to benchmark the serial implementation earlier
    - Obtain a basic comparison of both implementations
  - Test both implementations’ ability to solve much larger problems i.e. problem scaling
    - Much larger set of D, N, M values than the ones used in above step
  - Test how changing the number of vertical GPU threads used per block to compute $\hat{S}_{n+1}$ affects runtime performance
    - Same larger set of D, N, M values as step above

Parallel Implementation Performance Results

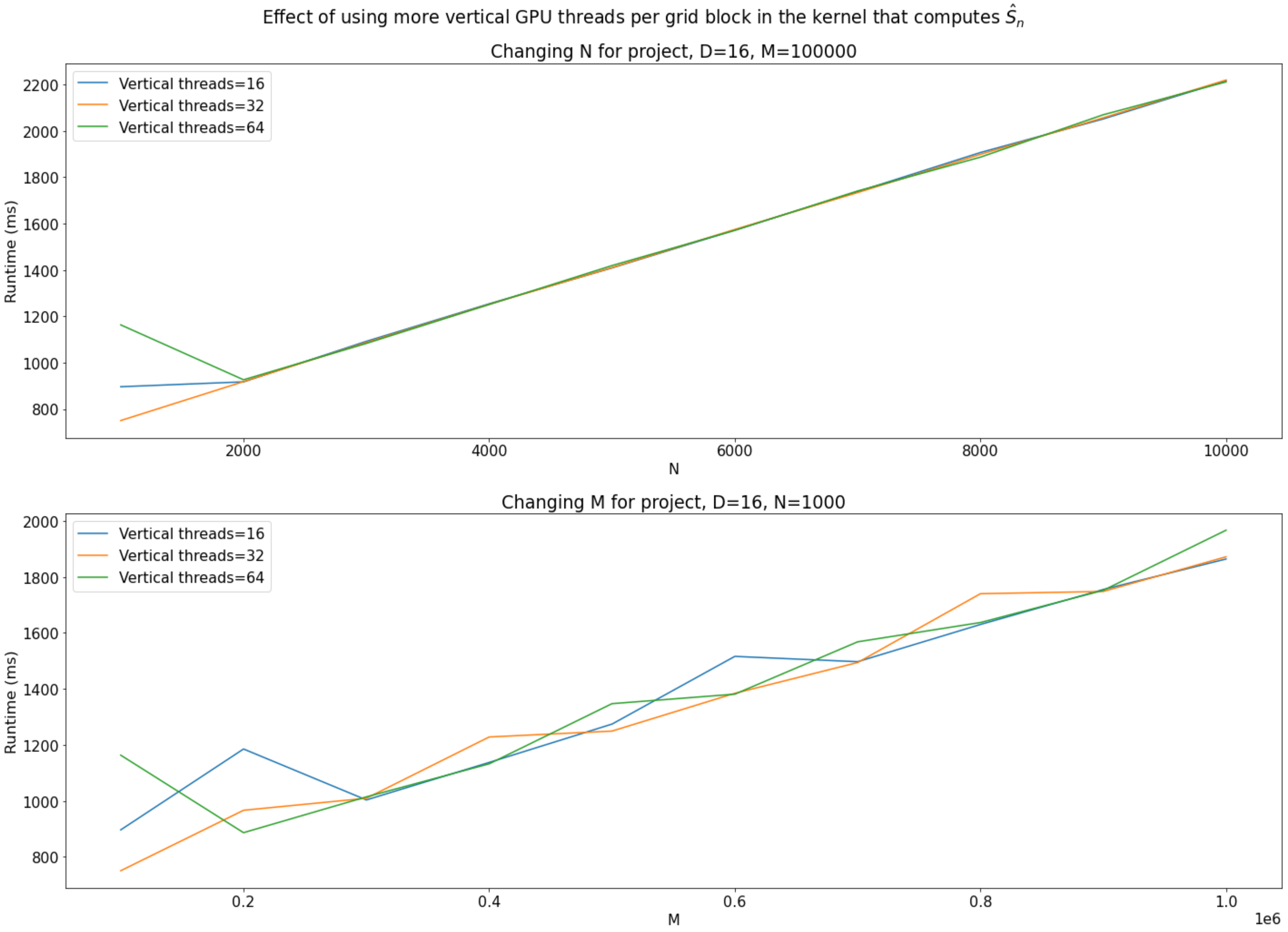


Parallel Implementation Performance Results





Parallel Implementation Performance Results



Reflection on Project

- Learnt C++!
  - Learnt to use Vim, how ssh works, NVIDIA CUDA in-depth
- Discovered the benefits and limitations of both implementations
  - Serial implementation does not actually scale well with very large problem sizes
    - Expected
  - Parallel implementation does
    - However, took much longer to implement successfully, debug
    - Overall longer in raw program length and also harder to read due to CUDA's strange function/method names
- In hindsight, should have used CUDA $memcheck$ to detect out of bounds memory access errors
  - Occurred several times in the parallel implementation and was very painful to debug e.g. which matrix/vector is causing runtime crash?

Reflection on COSC7502

- Overall, well-equipped with knowledge in various elements of HPC:
    - CPU vs. GPU architecture
    - Techniques such as manual and compiler optimisations, profiling
    - Programming models – AVX, OpenMP, MPI, CUDA, Hybrid
- Lack practical experience in MPI and Hybrid
    - Never attempted them in Assignment 2
    - Didn’t use them for the project either