Comparison of Direct and Iterative Solvers for Finite-Difference GPU-Based Valuation of American Options

Ilya Pogorelov, Paul Mullowney, Peter Messmer
(Tech-X Corporation)
Andrey Itkin and Lewis Biscamp
(Chicago Trading Company)

2009 SIAM Annual Meeting, July 6-10, Denver, Colorado
Introduction and Motivation

• Modern option trading systems require highly efficient and accurate algorithms for computing theoretical values and Greeks of American options.

• This requirement translates into solving concurrently multiple PDEs under very challenging time and hardware constraints.

• Recent advances in GPU computing allowed us to address this problem in a highly efficient and cost-effective way.

• We present results for one direct (cyclic reduction) and one iterative (biconjugate gradient) solver for the banded linear system resulting from discretization of the Black-Scholes-Merton equation.
Finite Difference-Based Pricing

- Assume the underlying asset price $S(t)$ undergoes geometric Brownian motion
  \[ dS(t) = S(t)[(r - \delta)dt + \sigma dW] \]

- American options can be exercised at a previously agreed-upon "strike price" $K$, at any time prior to maturity $T$ (discrete simulation/exercise times $t = 0, \ldots, T$ in FD models)

- Theoretical value $f$ of a put option written on a non-dividend-paying stock satisfies a parabolic PDE (Black-Scholes-Merton):
  \[ \frac{\partial f}{\partial t} + rS \frac{\partial f}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} = rf \]

- Solving backwards in time, the “initial” condition at $t = T$ is $f(t,S) = max(0, K-S)$ for both European and American puts
FD-Based Pricing (continued)

- Domain boundaries are chosen at $S = S_{max}$ such that $f(t,S_{max}) = 0$, and at $S = S_{min} = 0$ where $f(t,S_{min}) = K$ for an American put and $f(t,S_{min}) = K \exp(-r(T-t))$ for a European put.
- Implicit discretization schemes result in band-diagonal system solves at every timestep.
- Guaranteed stability.
- We studied performance on a GPU of cyclic reduction and biconjugate gradient solvers for the tridiagonal system resulting from an $O(\Delta s^2,\Delta t)$ discretization scheme.
- A very large number of systems have to be solved simultaneously, each for a different choice of the option parameters $K, T, r, \sigma$. 
Problem Description

- In practice, a very large number of linear systems result from a parameter space “sweep”
- For each underlying name, a (small) number of maturities $T$ and multiple strike prices $K$ are considered
- Varying $K$ modifies initial and boundary conditions, but not the matrix
- In addition, computation of Greeks requires re-computing the theoretical values of the options with new values of $r$ and $\sigma$
- This yields $N$ concurrent linear system solves per timestep, where $N$ is the number of points in parameter space
- A task-farming-based parallelism, ideally suited for GPU-based computing
Tech-X Corporation’s GPULib

- **GPULib**: an easy-to-use software library for computation acceleration using Graphics Processing Units (GPUs)
- GPULib provides a large collection of GPU vector operations in Very High Level languages

**Diagram:**
- MATLAB, IDL, and Python
- GPULib wrappers (language specific, includes software emulator)
  - GPULib functions: Vector Arithmetic, Data Manipulation, Complex Operations
  - NVIDIA functions: cuBLAS, cuFFT
  - CUDA Runtime API
- GPU
GPULib (cont-d)

- GPU viewed as co-processor
- Explicit data transfer to/from GPU
- Interface to GPU matches language style
- Performance via vector operations on GPU
- No need for low-level code development
- Supports CUDA enabled devices
In addition to cuBLAS and cuFFT, GPULib has a broad set of:
- Vector arithmetic operations
- Reductions (e.g., dot products)
- Interpolation operations
- Array reshaping
- Advanced physics algorithms
- Mersenne Twister RNG, …

Vector Kernel Performance

CPU: Intel Core2 6400 (dual core), 2.1 GHz, 3 GB RAM, GPU: NVIDIA GeForce 8800 GTX (128 processing elements)
Implementing Linear Solvers on GPUs

- Test system: NVIDIA TESLA C1060 GPU
- Computation strategy: one thread-block per linear system solve
- Each GPU thread-block has 256 threads and 16k shared memory
- Systems are limited to 256 equation so as to keep temporary data in shared memory (both CR and BiCG)
- Each thread effectively handles one ‘row’ in the vector arithmetic
Implementing Linear Solvers on GPUs (continued)

• We implemented the cyclic reduction and biconjugate gradient algorithms
• Algorithms are implemented in single precision on the GPU
• One of the main issues: efficient reduction operations (e.g., dot product) in BiCG require many thread synchronization steps
• Several reduction operations per iteration
• Reductions are likely the main source of slowdown in BiCG
Algorithm Performance

- Test system: NVIDIA TESLA C1060 GPU
- Up to \( N = 5000 \) systems (points in parameter space in our sweeps)
- We expect comparable results for parameter sweeps with different matrices
- Figure to the left shows the speedup of the GPULib cyclic reduction and biconjugate gradient solvers versus MATLAB’s \( \text{A\backslash b} \) solve on the 2.4GHz Intel Core 2 Duo CPU
- CR shows 1000X and BiCG shows 50X acceleration for \( N = 5000 \)

The speedup of the GPULib cyclic reduction and biconjugate gradient solvers vs MATLAB’s \( \text{A}\backslash \text{b} \) solve. NVIDIA TESLA C1060 GPU and 2.4 GHz Intel Core 2 Duo used for these tests.