Lecture 4

- Finish the Hull-White Monte Carlo examples from Lecture 3

- Running a finite difference solver on a GPU

- Suggested sections from John Hull, *Option, Futures, and Other Derivatives* 9th ed. on finance applications of finite difference methods
    - Chapter 21, Basic numerical procedures, especially section 21.8 on Finite difference methods
    - Chapter 13, Binomial trees
    - Chapter 26, Exotic options
    - Chapter 27, More on models and numerical procedures

- One more problem set to apply finite difference methods on GPU's

Lecture 4 – Running Finite Difference Solvers on GPU's

- First, a review of finite difference methods.

- Many of the pricing problems in Finance can be solved numerically by solving the partial differential equation that the valuation or pricing function must satisfy. In fact, finite difference is the best solution method for some problems, such as pricing American or Bermudan options.

- Start with the Black-Scholes model. The pricing function must satisfy the following PDE.

$$0 = \frac{\partial f}{\partial t} + (r-\delta)\, S \frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} - r f$$

- In the implicit finite difference method, we solve the model on a grid (i, j) as follows

$$\frac{\partial f}{\partial S} = \frac{f_{i,j+1} - f_{i,j-1}}{2\,\Delta S} \qquad \frac{\partial^2 f}{\partial S^2} = \frac{f_{i,j+1} + f_{i,j-1} - 2 f_{i,j}}{\Delta S} \qquad \frac{\partial f}{\partial t} = \frac{f_{i+1,j} - f_{i,j}}{\Delta t}$$

- i is the index for the time dimension and j is the index for the stock price increments

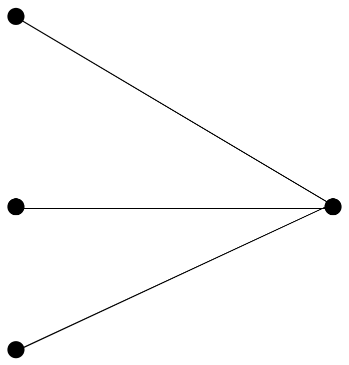
Lecture 4 – Running Finite Difference Solvers on GPU’s (cont.)

- In the explicit finite difference method, we solve the derivatives on the state variable differently in forward time

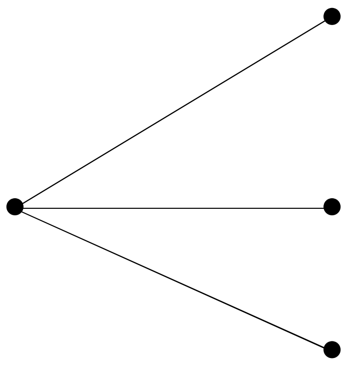
$$\frac{\partial f}{\partial S} = \frac{f_{i+1,j+1} - f_{i+1,j-1}}{2\,\Delta S} \qquad \frac{\partial^2 f}{\partial S^2} = \frac{f_{i+1,j+1} + f_{i+1,j-1} - 2 f_{i+1,j}}{\Delta S} \qquad \frac{\partial f}{\partial t} = \frac{f_{i+1,j} - f_{i,j}}{\Delta t}$$

- The difference between the two methods can viewed as follows

Implicit method



Explicit method



Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- The explicit finite difference method can be faster to calculate, but it is not guaranteed to be stable. The variance needs to be constant so that one can fix the ratio between Δt and Δx, which is the grid size for the state variable. In the Black-Scholes model, we can make the explicit method stable by transforming from the stock price S to a new state variable $x = \log S$.

- The explicit solution for Black-Scholes is not the binomial model, but it is close. In fact, the explicit method for Black-Scholes produces the same results and algorithm as the lattice or trinomial model, which is based on probabilities over small time steps.

- The implicit method is popular because it is guaranteed to be stable, and it is necessary for many models.

- Boundary conditions are also important and necessary for solving models with finite difference methods. For option pricing, we impose the boundary condition at expiration, namely that the valuation function must equal the payoff at expiration. Other boundary conditions may be a minimum of zero for the price, and there are maximum valuations for options. For American options, there is an additional boundary condition that the valuation must be at least as much as the intrinsic value which is imposed by early exercise.

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- An example with a stochastic volatility model

$$d \log S(t) = [r(t) - \delta(t) - \tfrac{1}{2} v(t)]\, dt + \sqrt{v(t)}\, dz_1$$

$$v(t) = \exp(y(t))$$

$$dy(t) = \kappa\,[\,\theta - y(t)\,]\, dt + \sigma\, dz_2$$

- This is the SV model with a mean reverting lognormal process for the stochastic variance, covered back in Lecture 3. There are no known closed form solutions for option prices in this model. I have chosen this model as an example because there is some empirical evidence that stock price volatility is closer to a lognormal process, versus the square root process in the Heston model.

- In the Heston model, the square root process is used for stochastic variance

$$d\, v(t) = \kappa\,[\,\theta - v(t)\,]\, dt + \sigma \sqrt{v(t)}\, dz_2$$

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- The PDE for the lognormal SV model

$$0 = \frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial (\log P)^2} e^{y} + \frac{1}{2}\frac{\partial^2 V}{\partial y^2}\sigma^2 + \rho\sigma\, e^{0.5y}\, \frac{\partial^2 V}{\partial \log P\, \partial y}$$

$$+ \frac{\partial V}{\partial \log P}[r(t) - \delta(t) - \tfrac{1}{2}e^{y}] + \frac{\partial V}{\partial y}\kappa[\theta - y] - r(t)V$$

- To solve models like this, we typically do a transformation to remove the cross term in the second derivatives. For the lognormal model,

$$x = \log P - \frac{2\rho\, e^{0.5y}}{\sigma}$$

- And we can use Ito's lemma to derive *dx*: $dx = d\,\log P - \frac{\rho\, e^{0.5y}}{\sigma} dy - \tfrac{1}{4}\rho\, e^{0.5y}\sigma\, dt$

- Transformation for Heston model: $x = \log P - {}^{\rho\, v}\!/_{\sigma}$

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- We can verify that $E(dx\,dy) = 0$
- The we need to work out the details for the PDE that applies to *x* and *y*
- Or we can apply the change of variable to the PDE. Both methods should produce the same result.
- We are transforming from a function *V* to a function *f* $\quad V(t, \log P, y) = f(t, x, y)$

$$\frac{\partial V}{\partial \log P} = \frac{\partial f}{\partial \log P} = \frac{\partial f}{\partial x} \qquad \frac{\partial V}{\partial t} = \frac{\partial f}{\partial t}$$

$$\frac{\partial V}{\partial y} = \frac{\partial f}{\partial y} - \frac{\partial f}{\partial x}\frac{\rho e^{0.5y}}{\sigma} \qquad \frac{\partial^2 V}{\partial (\log P)^2} = \frac{\partial^2 f}{\partial (\log P)^2} = \frac{\partial^2 f}{\partial x^2}$$

$$\frac{\partial^2 V}{\partial (\log P)\,\partial y} = \frac{\partial^2 f}{\partial x\,\partial y} - \frac{\partial^2 f}{\partial x^2}\frac{\rho e^{0.5y}}{\sigma} \qquad \frac{\partial^2 V}{\partial y^2} = \frac{\partial^2 f}{\partial y^2} - 2\frac{\partial^2 f}{\partial x\,\partial y}\frac{\rho e^{0.5y}}{\sigma} + \frac{\partial^2 f}{\partial x^2}\frac{\rho^2 e^{y}}{\sigma^2} - \tfrac{1}{2}\frac{\partial f}{\partial x}\frac{\rho e^{0.5y}}{\sigma}$$

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- The PDE for x and y is

$$0 = \frac{\partial f}{\partial t} + \frac{1}{2}\frac{\partial^2 f}{\partial x^2} e^{y}\left(1-\rho^2\right) + \frac{1}{2}\frac{\partial^2 f}{\partial y^2}\sigma^2$$

$$+ \frac{\partial f}{\partial x}\left[r(t) - \delta(t) - \tfrac{1}{2}e^{y} - \frac{\rho\, e^{0.5y}\left(\kappa\theta - \kappa y + \tfrac{1}{4}\sigma^2\right)}{\sigma}\right] + \frac{\partial f}{\partial y}\kappa[\theta - y] - r(t)\, f$$

- We have succeeded in removing the cross term and now we need to apply a finite difference method for 2 state variables and time

- With the stochastic volatility, we need to solve the x variable, which contains the stock price, with the implicit method. We can solve the y variable with the explicit or implicit method. If we solve y with the implicit method, we would be using ADI, or alternating direction implicit. I will solve the y variable explicitly. If done properly, it will be stable and is actually marginally faster than the implicit method. This is possible because I have assumed that σ is constant.

- And I will break the problem into 3 steps. This is also known as time splitting.

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- Define $t,\ t' = t + \Delta t/3,\ \ t'' = t + 2\Delta t/3,\ \ t''' = t + \Delta t$ and $x_i = i\,\Delta x$ and $y_j = j\,\Delta y$

- We use the following approximations for the partial derivatives

$$\frac{\partial f}{\partial t} = \frac{1}{3}\left\{ \frac{f(t+\Delta t/3, x_i, y_j) - f(t, x_i, y_j)}{\Delta t/3} + \frac{f(t+2\Delta t/3, x_i, y_j) - f(t+\Delta t/3, x_i, y_j)}{\Delta t/3} \right.$$

$$\left. + \frac{f(t+\Delta t, x_i, y_j) - f(t+2\,\Delta t/3, x_i, y_j)}{\Delta t/3} \right\}$$

$$= \frac{f(t', x_i, y_j) - f(t, x_i, y_j)}{\Delta t} + \frac{f(t'', x_i, y_j) - f(t', x_i, y_j)}{\Delta t} + \frac{f(t''', x_i, y_j) - f(t'', x_i, y_j)}{\Delta t}$$

$$\frac{\partial f}{\partial y} = \frac{f(t'', x_i, y_{j+1}) - f(t'', x_i, y_{j-1})}{2\Delta y}$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{(f(t''', x_i, y_{j+1}) - f(t''', x_i, y_j)) - (f(t''', x_i, y_j) - f(t''', x_i, y_{j-1}))}{\Delta y^2}$$

$$= \frac{f(t''', x_i, y_{j+1}) + f(t''', x_i, y_{j-1}) - 2\,f(t''', x_i, y_j)}{\Delta y^2}$$

Lecture 4 – Running Finite Difference Solvers on GPU’s (cont.)

$$\frac{\partial f}{\partial x} = \frac{f(t, x_{i+1}, y_j) - f(t, x_{i-1}, y_j)}{2\Delta x}$$

$$\frac{\partial^2 f}{\partial x^2} = \frac{f(t, x_{i+1}, y_j) + f(t, x_{i-1}, y_j) - 2\, f(t, x_i, y_j)}{\Delta x^2}$$

- We plug these derivatives into the PDE $\rightarrow$ go to next slide

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- Resulting PDE to be solved numerically

$$
\begin{aligned}
0 \;=\; & [\,f(t',x_i,y_j) - f(t,x_i,y_j)] + \tfrac{1}{2} e^{y_i}\left(1-\rho^2\right)\frac{\Delta t}{\Delta x^2}[\,f(t,x_{i+1},y_j) + f(t,x_{i-1},y_j) - 2\,f(t,x_i,y_j)] \\
& + \left(r(t) - \delta(t) - \tfrac{1}{2} e^{y_i} - \frac{\rho\, e^{0.5y}\left(\kappa\theta - \kappa y + \tfrac{1}{4}\sigma^2\right)}{\sigma} \right)\frac{\Delta t}{2\Delta x}[\,f(t,x_{i+1},y_j) - f(t,x_{i-1},y_j)] \\
& - r(t)\Delta t\; f(t,x_i,y_j) \\
& + \left[\,f(t'',x_i,y_j) - \; f(t',x_i,y_j)\right] + \max[0,\,(\kappa\theta - \kappa y_j)]\frac{\Delta t}{\Delta y}[\,f(t'',x_i,y_{j+1}) - f(t'',x_i,y_j)] \\
& + \; \min[0,\,(\kappa\theta) - \kappa y_j)]\frac{\Delta t}{\Delta y}[\,f(t'',x_i,y_j) - f(t'',x_i,y_{j-1})] \; + \; [\,f(t''',x_i,y_j) - f(t'',x_i,y_j)] \\
& + \; \tfrac{1}{2}\sigma^2 \frac{\Delta t}{\Delta y^2}[\,f(t''',x_i,y_{j+1}) + f(t''',x_i,y_{j-1}) - 2\,f(t''',x_i,y_j)]
\end{aligned}
$$

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- Solve this PDE sequentially as follows in 3 steps

- Step 1: $f(t'', x_i, y_j) =$

$$[f(t''', x_i, y_{j+1}) + f(t''', x_i, y_{j-1})]\tfrac{1}{2}\sigma^2 \frac{\Delta t}{\Delta y^2} \;+\; f(t''', x_i, y_j)\left(1 - \sigma^2 \frac{\Delta t}{\Delta y^2}\right)$$

- Step 2: $f(t', x_i, y_j) \;=\; f(t'', x_i, y_j) + \max[0,\ \kappa\theta - \kappa y_j]\dfrac{\Delta t}{\Delta y}\left[f(t'', x_i, y_{j+1}) - f(t'', x_i, y_j)\right]$

$$+\;\min[0, \kappa\theta - \kappa y_j]\frac{\Delta t}{\Delta y}[f(t'', x_i, y_j) - f(t'', x_i, y_{j-1})]$$

- Step 3: $f(t, x_i, y_j)\left(1 + r(t)\Delta t + e^{y_i}\left(1-\rho^2\right)\dfrac{\Delta t}{\Delta x^2}\right)$

$$+\; f(t, x_{i+1}, y_j)\left(-\tfrac{1}{2}e^{y_i}\left(1-\rho^2\right)\frac{\Delta t}{\Delta x^2} - [r(t) - \delta(t) - \tfrac{1}{2}e^{y_i} - \frac{\rho\, e^{0.5y}\left(\kappa\theta - \kappa y + \frac{1}{4}\sigma^2\right)}{\sigma}]\frac{\Delta t}{2\Delta x}\right)$$

$$+\; f(t, x_{i-1}, y_j)\left(-\tfrac{1}{2}e^{y_i}\left(1-\rho^2\right)\frac{\Delta t}{\Delta x^2} + [r(t) - \delta(t) - \tfrac{1}{2}e^{y_i} - \frac{\rho\, e^{0.5y}\left(\kappa\theta - \kappa y + \frac{1}{4}\sigma^2\right)}{\sigma}]\frac{\Delta t}{2\Delta x}\right)$$

$$=\; f(t', x_i, y_j)$$

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- The right hand side of the first equation on the previous slide is the explicit solution on the second derivatives of *y*. We solve the mean reversion part of *y* in the second step. These 2 parts of the equation can be solved together. I find it easier and convenient to split these two parts of the equation. The last step is the implicit solution on *x*.

- For the explicit solution on *y*, we need to use the mesh ratio to guarantee stability

$$0 < \sigma^2 \frac{\Delta t}{\Delta y^2} < 1 \quad \rightarrow \quad \Delta y = \sigma \sqrt{1.5\Delta t}$$

- The first part of the explicit solution has the following form. $(0 < 2a < 1\,,\ t'' < t''')$

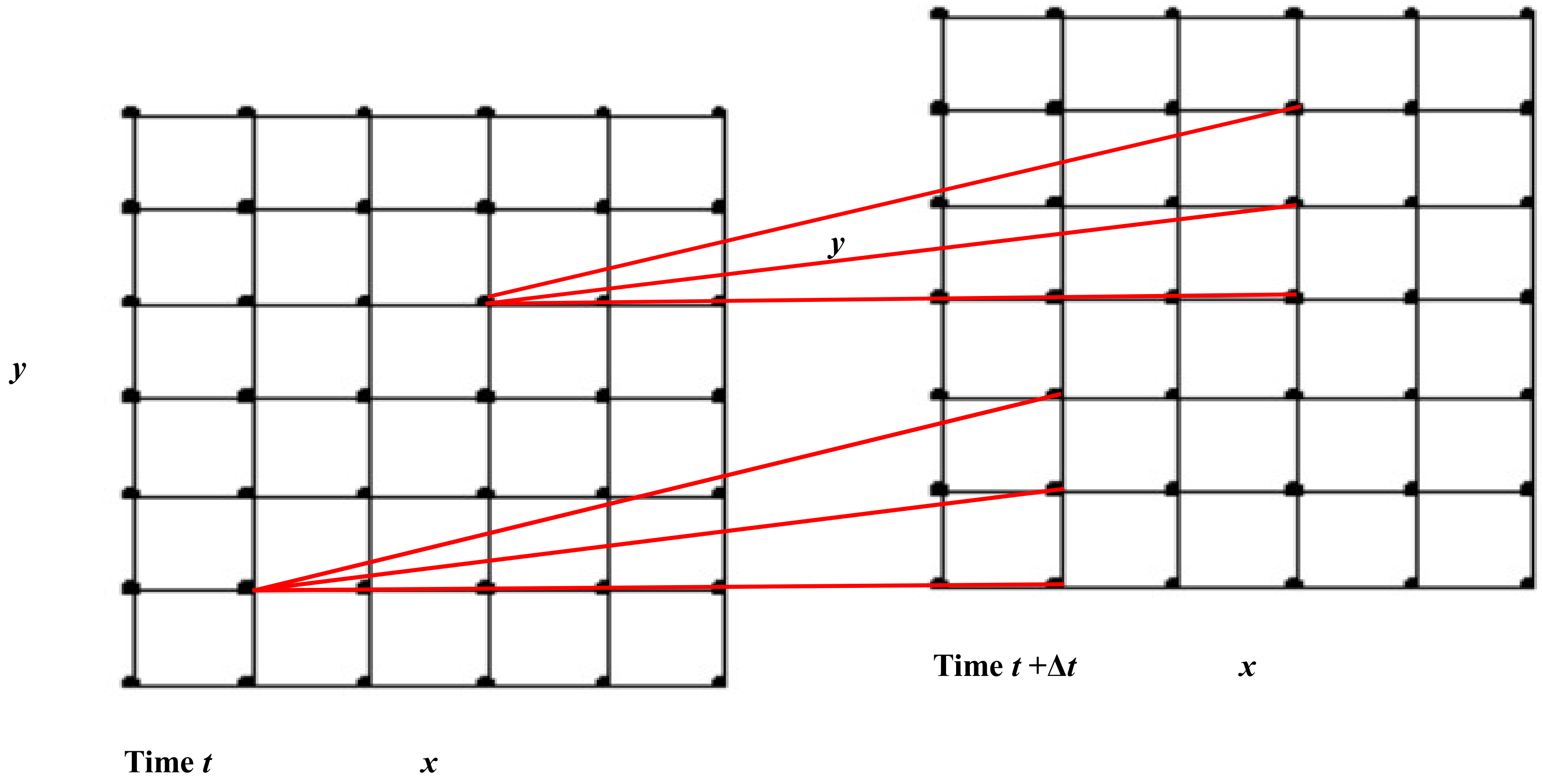
$$f\left(t'', x_i, y_j\right) = a\, f\left(t''', x_i, y_{j+1}\right) + (1-2a) f(t''', x_i, y_j) + a\, f(t''', x_i, y_{j-1})$$

  and we write a loop on *i* and *j* to perform to perform this calculation at each grid point.

- The next slide shows a graphical representation of this calculation on the finite difference grid.

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- A view of section of the grid between 2 time steps: explicit solution method in *y* dimension



Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- The second step in the calculation handles the mean reversion in the *y* state variable.

$$f\left(t',x_i,y_j\right) = f\left(t'',x_i,y_j\right) + \max[0,\ \kappa\theta-\kappa y]\frac{\Delta t}{\Delta y}\left(f(t'',x_i,y_{j+1}) - f(t'',x_i,y_j)\right)$$
$$+\min[0,\kappa\theta-\kappa y]\frac{\Delta t}{\Delta y}\left(f\left(t'',x_i,y_j\right) - f\left(t'',x_i,y_{j-1}\right)\right)$$

- Similar to the calculation in step 1, we write a loop across *i* and *j* to perform these calculations.

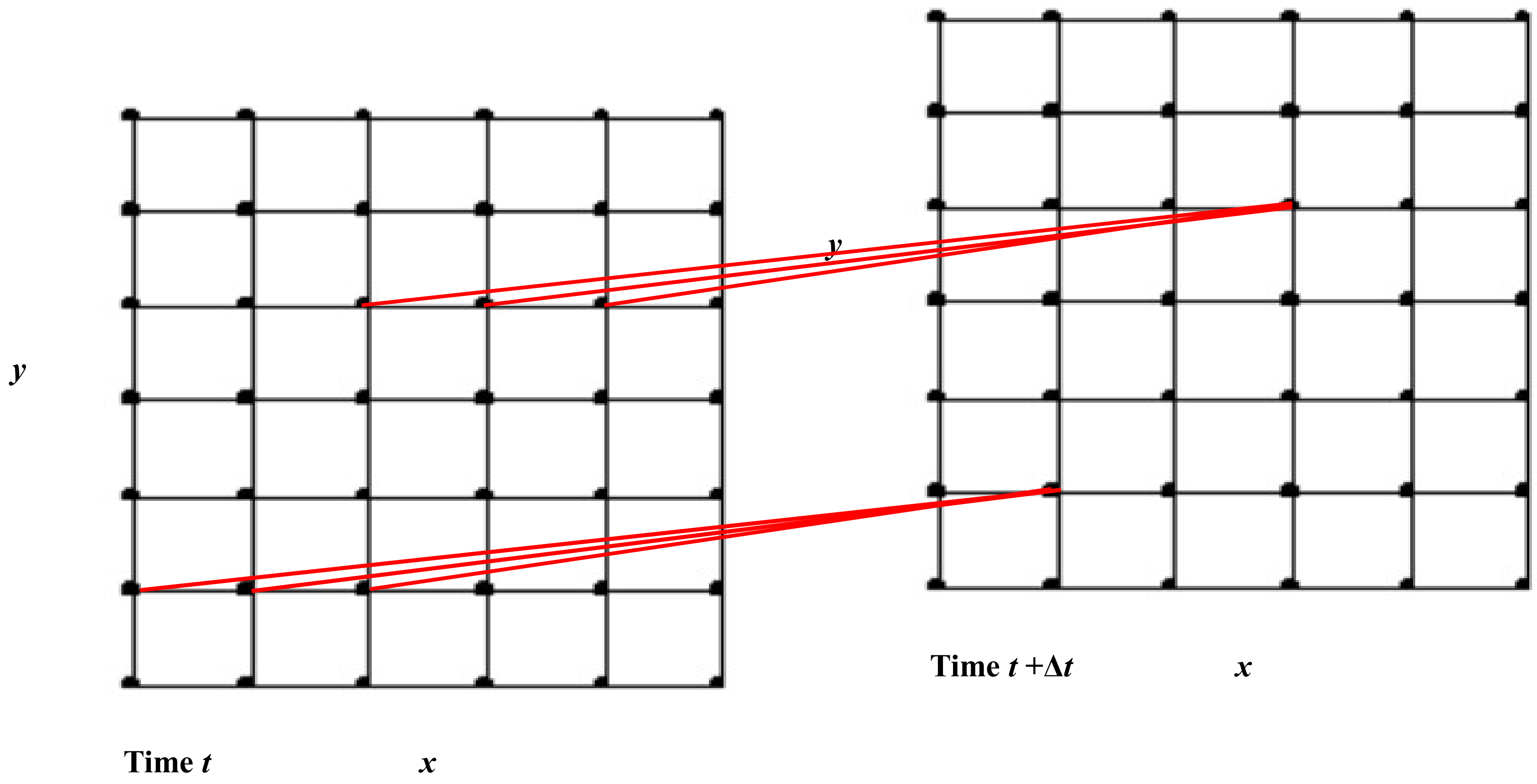
- The last step, step 3 is more complicated. This is the implicit solution, which has the following form.

$$a_i\, f\left(t,x_{i-1},y_j\right) + b_i f\left(t,x_i,y_j\right) + c_i f\left(t,x_{i+1},y_j\right) = f(t',x_i,y_j)$$

- The solution method is shown graphically on the next slide

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- A view of section of the grid between 2 time steps: implicit solution method in *x* dimension



Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- Boundary conditions must be imposed at $i = 0$ and at at $i = n$, the extreme points in the x dimension. For call and put options, we impose a boundary condition that the second derivative with respect to the stock price is going to zero at the minimum stock price (minimum of x) and at the maximum stock price (maximum x). With the boundary conditions, the implicit solution can be represented as the following solution which requires a matrix inversion.

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

- The matrix is tridiagonal and can be inverted quickly using a recursive method. I use Thomas's algorithm for inverting this matrix for the implicit solution.
  See https://www.cfd-online.com/Wiki/Tridiagonal_matrix_algorithm_-_TDMA_(Thomas_algorithm)

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- To impose the boundary conditions, go back to the PDE for *x* and *y*

$$0 = \frac{\partial f}{\partial t} + \frac{1}{2}\frac{\partial^2 f}{\partial x^2} e^y (1-\rho^2) + \frac{1}{2}\frac{\partial^2 f}{\partial y^2}\sigma^2$$

$$+ \frac{\partial f}{\partial x}\left(r(t) - \delta(t) - \frac{1}{2}\, e^y - \frac{\rho e^{0.5y}\left(\kappa\theta - \kappa y + \frac{1}{4}\sigma^2\right)}{\sigma}\right) + \frac{\partial f}{\partial y}\, \kappa(\theta - y) - r(t) f$$

- $\frac{\partial f}{\partial S} = \frac{\partial V}{\partial x}\frac{1}{S}$ and $\frac{\partial^2 f}{\partial s^2} = \frac{\partial^2 f}{\partial x^2}\frac{1}{s^2} - \frac{\partial V}{\partial x}\frac{1}{s^2} = 0$ at the boundaries of *x*. $\frac{\partial^2 f}{\partial x^2} = \frac{\partial V}{\partial x}$

- At the boundaries of *x*, we get the following PDE

$$0 = \frac{\partial f}{\partial t} + \frac{1}{2}\frac{\partial^2 f}{\partial y^2}\sigma^2 + \frac{\partial f}{\partial x}\left(r(t) - \delta(t) - \frac{1}{2}\, \rho e^y - \frac{\rho e^{0.5y}\left(\kappa\theta - \kappa y + \frac{1}{4}\sigma^2\right)}{\sigma}\right)$$

$$+ \frac{\partial f}{\partial y}\, \kappa(\theta - y) - r(t) f$$

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- At the minimum for the stock price, *i* = 0, approximate the partial derivative with respect to *x* as follows

$$\frac{\partial f}{\partial x} = \frac{f(t, x_2, y_j) - f(t, x_1, y_j)}{\Delta x}$$

$$\left(1 + r\Delta t + \mu \frac{\Delta t}{\Delta x}\right) f(t, x_1, y_j) - \mu \frac{\Delta t}{\Delta x} f(t, x_2, y_j) = f(t', x_1, y_j)$$

with $\mu = r(t) - \delta(t) - \frac{1}{2}\rho e^{y} - \frac{\rho e^{0.5y}\left(\kappa\theta - \kappa y + \frac{1}{4}\sigma^2\right)}{\sigma}$

- At the maximum for the stock price, *i* = *n*, approximate the partial derivative with respect to *x* as follows

$$\frac{\partial f}{\partial x} = \frac{f(t, x_n, y_j) - f(t, x_{n-1}, y_j)}{\Delta x}$$

$$\left(1 + r\Delta t - \mu \frac{\Delta t}{\Delta x}\right) f(t, x_n, y_j) + \mu \frac{\Delta t}{\Delta x} f(t, x_{n-1}, y_j) = f(t', x_1, y_j)$$

The forward sweep consists of modifying the coefficients as follows, denoting the new coefficients with primes:

$$c'_i = \begin{cases} \dfrac{c_i}{b_i} & ; \quad i = 1 \\ \dfrac{c_i}{b_i - a_i c'_{i-1}} & ; \quad i = 2, 3, \dots, n-1 \end{cases}$$

and

$$d'_i = \begin{cases} \dfrac{d_i}{b_i} & ; \quad i = 1 \\ \dfrac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}} & ; \quad i = 2, 3, \dots, n. \end{cases}$$

The solution is then obtained by back substitution:

$$x_n = d'_n$$
$$x_i = d'_i - c'_i x_{i+1} \qquad ; \, i = n-1, n-2, \dots, 1.$$

The method above preserves the original coefficient vectors. If this is not required, then a much simpler form of the algorithm is

$$\left. \begin{aligned} w_i &= \frac{a_i}{b_{i-1}} \\ b_i &:= b_i - w_i c_{i-1} \\ d_i &:= d_i - w_i d_{i-1} \end{aligned} \right., i = 2, 3, \dots, n$$

followed by the back substitution

$$x_n = \frac{d_n}{b_n}$$
$$x_i = \frac{d_i - c_i x_{i+1}}{b_i}, i = n-1, n-2, \dots, 1$$

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- I use Thomas's algorithm for inverting the matrix for the implicit solution. See https://www.cfd-online.com/Wiki/Tridiagonal_matrix_algorithm_-_TDMA_(Thomas_algorithm)_ for the method and the algorithm

- There are boundary that need to be imposed at the minimum and maximum of x. For option prices, puts and calls, as we approach the extreme stock prices, the second derivative with respect to the stock price is going to zero. The function is becoming linear. What about the function with respect to log P or x. In these cases, the second derivative condition means that the second derivative is equal to the first derivative with respect to either log P or x, depending on how we build the model.

- Discussion of dividends and dividend yields
  - Single discrete dividend that has been declared: subtract PV(Dividend) from current stock price
  - Dividend yield: if it is deterministic, then it can be handled in the drift as a reduction of the interest rate.
  - Future discrete dividends as a percentage or yield on stock price.

- In problem set 3, you need to code a 2 factor SV model to price American options.

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- Discussion of dividends and dividend yields

  1 - For a single discrete dividend that has been declared: subtract PV(Dividend) from current stock price

  2 - Dividend yield: if it is deterministic, then it can be handled in the drift as a reduction of the interest rate.

  3 - Future discrete dividends as a percentage or yield on stock price.

- In the models that I code, I distinguish between trading days and non-trading days. I run the model above for trading days, then assume no volatility and no change in *y* on weekends and holidays. With this assumption, the PDE simplifies for non-trading days.

$$f(t,x_i,y_j) = \frac{1}{1+r\Delta t}\left\{ f(t',x_i,y_j) + [r(t)-\delta(t)]\frac{\Delta t}{2\Delta x}\left(f(t',x_{i+1},y_j) - f(t',x_{i-1},y_j) \right)\right\}$$

- For problem sets and the course project, you are not required to distinguish between trading days and non-trading days. And you are not required to handle the impact of dividends. These are features that equity trading desks incorporate in their models.

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- I recommend implementing the SV model the CPU first. It is easier to test and debug. Then copy your code for the finite difference algorithm GPU device functions, to run on GPU.

- The finite difference algorithm must be solved sequentially by starting at expiration and working backwards. This part cannot be executed in parallel. But we can run the calculations for each of the 3 steps in parallel, and it is very easy to move the explicit solution and the implicit solution to the GPU. Set up a function to do the finite difference algorithm. On the CPU, set up a loop to go backwards through time sequentially. The explicit and implicit solutions will require looping on the 2 dimensional grid to do the calculations. Write a function to do each one of these solution methods, with the loops inside the function. For the explicit solution, you loop over x_i and y_j. The recursive method for the implicit solution must be calculated sequentially. There is a loop over y_j and the j loop can be calculated in parallel. Note that for each y_j, we are calculating an implicit solution sequentially. On the GPU, you must also allocate an additional matrix for workspace to be able to perform the parallel calculations. See example.

Lecture 4 – Running Finite Difference Solvers on GPU's (cont.)

- The 3 factor Hull-White m odel. Price American Eurodollar futures options.

$$0 = \frac{\partial f}{\partial t} + \frac{1}{2}\sigma_0^2 \frac{\partial^2 f}{\partial r^2} + \frac{1}{2}\sigma_1^2 \frac{\partial^2 f}{\partial y_1^2} + \frac{1}{2}\sigma_2^2 \frac{\partial^2 f}{\partial y_2^2} + \kappa_0 (y_1 + y_2 - r)\frac{\partial f}{\partial r}$$

$$+ \kappa_1(\theta_1 - y_1)\frac{\partial f}{\partial y_1} - \kappa_2 y_2 \frac{\partial f}{\partial y_2} - rf$$

- Use the explicit method to solve in the y_1 and y_2 dimensions. Use the implicit method to solve in the *r* dimension.

- Go to examples for CPU code and GPU code $\rightarrow$