Lecture 5

- Topics for today
  - Finish Lecture 4 on finite difference methods; do examples with Hull-White model
  - Second, go over methods for doing parallel processing with python
    - run Cuda functions from python using pycuda
    - concurrent.futures in python
    - calling C++/Cuda functions, wrapped in a dll, in from python using ctypes
  - See online python documentation for pycuda, concurrent.futures, and ctypes

    https://documen.tician.de/pycuda/

    https://docs.python.org/3/library/concurrent.futures.html

    https://docs.python.org/2/library/ctypes.html

Prototyping and Developing GPU Accelerated Solutions with Python and CUDA

Luciano Martins

Principal Software Engineer
Amazon Web Services

Robert Sohigian

Technical Marketing Engineer

NVIDIA

2018-04-12 Rev 6

Agenda

- Introduction to Python
- GPU-Accelerated Computing
- CUDA
- Why use Python with GPUs?
- Accelerating Python
- Comparing Results With/Without GPU Support
- Summary
- Documentation

Why use Python with GPUs?

- Interpreted languages are slow for high performance needs
- Python needs assistance for those tasks
- Keep the best of both scenarios:
  - Quick development and prototyping with Python
  - Use high processing power and speed of GPU
- Can deliver quick results for complex projects
- Gives a business decision choice at the end

Accelerating Python

- GPU + Python projects are arising every day
- Accelerated code may be pure Python or adding C-code
- Focusing here on the following modules
  - PyCUDA
  - Numba
  - cudamat
  - cupy
  - scikit-cuda

PyCUDA

- A Python wrapper to CUDA API
- Requires C programming knowledge (kernel)
- Gives speed to Python – near zero wrapping
- Compiles the CUDA code copy to GPU
- CUDA errors translated to Python exceptions
- Easy installation

```
root@hell:~# pip3 install pycuda
Collecting pycuda
  Downloading pycuda-2017.1.1.tar.gz (1.6MB)
    100% |████████████████████████████████| 1.6MB 963kB/s
```

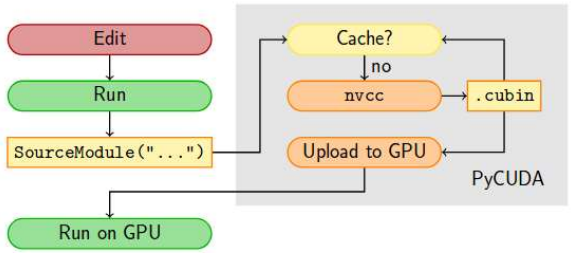
```
root@hell:~# pip3 show pycuda
Name: pycuda
Version: 2017.1.1
Summary: Python wrapper for Nvidia CUDA
Home-page: http://mathema.tician.de/software/pycuda
Author: Andreas Kloeckner
Author-email: inform@tiker.net
```

PyCUDA

Editor - /Downloads/GTC2018/GTC_sample.py

GTC_sample.py*

```
#!/usr/bin/env python

# importing modules
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy

# creating a numpy array
a = numpy.random.randn(4,4)

# maing this array as single precision format
a = a.astype(numpy.float32)

# allocating GPU memory to store the a array
a_gpu = cuda.mem_alloc(a.nbytes)

# copy the array to GPU memory
cuda.memcpy_htod(a_gpu, a)

# declaring the kernel to be run on the GPU
# using PyCUDA as C wrapper
mod = SourceModule("""
  __global__ void doublify(float *a)
  {
    int idx = threadIdx.x + threadIdx.y*4;
    a[idx] *= 2;
  }
  """)

# importing to the python realm the 'doublify' function
# from the kernel and passing the a_gpu as argument
func = mod.get_function("doublify")
func(a_gpu, block=(4,4,1))

# fetching the data back from the GPU and displaying it
# also showing the original a
a_doubled = numpy.empty_like(a)
cuda.memcpy_dtoh(a_doubled, a_gpu)
print(a_doubled)
print(a)
```



A simple example of python code that uses pycuda

```
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy

a = numpy.random.randn(4,4)
a = a.astype(numpy.float32)
a_gpu = cuda.mem_alloc(a.nbytes)
cuda.memcpy_htod(a_gpu, a)

mod = SourceModule("""
  __global__ void doublify(float *a)
  {
    int idx = threadIdx.x + threadIdx.y*4;
    a[idx] *= 2;
  }
  """)

func = mod.get_function("doublify")
func(a_gpu, block=(4,4,1))

a_doubled = numpy.empty_like(a)
cuda.memcpy_dtoh(a_doubled, a_gpu)
print a_doubled
print a


```

Summary

- PyCUDA
  - A CUDA Python wrapper
  - C code added directly on the Python project
  - All CUDA libraries support
  - Relevant complexity due to the kernels in C

Documentation

- Python
  https://www.python.org/doc/
- CUDA
  http://docs.nvidia.com/cuda/
- PyCUDA
  https://documen.tician.de/pycuda/
- Numba
  http://numba.pydata.org/doc.html

→ **Go to python examples with pycuda**

1) **the simple example of pycuda on slide 8**

1) **A simulation example using pycuda**

3) Examples using ctpyes to call C++/Cuda functions to run FX Stochastic Volatility model simulations in parallel on CPU and GPU

The strategy with ctypes is to use C++/Cuda to build a dll function that will do the intensive computing. The dll function can be called from python using ctypes

→ Go to Bloomberg to show market data for FX spot, forwards, and options

3) Examples using ctpyes to call C++/Cuda functions to run FX Stochastic Volatility model simulations in parallel on CPU and GPU (cont.)

Market data for FX spot, forwards, and options

Black-Scholes mode for European FX calls

$$C(S(t), t; T, K) = S(t)\, N(d_1) - e^{-r_d(T-t)}\, K\, N(d_2)$$

$$d_1 = \frac{\log\left(S/K\right) + \left(r_d - r_f + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

FX Forward rate: $F = S\exp\left((r_d - r_f)(T-t)\right)$ or $\log F = \log S + (r_d - r_f)(T-t)$

Strikes at-the money forward, plus $\Delta = N(d_1) = 0.50, 0.25, 0.10$ (and 0.90, 0.75)

Put $\Delta = 1 -$ Call $\Delta = 1 - N(d_1)$, $N(d_1) = 0.90$ corresponds to a put $\Delta = 0.10$

3) Examples using ctpyes to call C++/Cuda functions to run FX Stochastic Volatility model simulations in parallel on CPU and GPU (cont.)

Calculate the strikes corresponding to the different Δ's

For $d_1 = \frac{\log(S/K)+\left(r_d - r_f + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}$, find K so that $N(d_1)$ = 0.50, 0.25, 0.10, …

Solving for $N(d_1) = x$, $\log K = \log F + \frac{1}{2}\sigma^2(T-t) - \sigma\sqrt{T-t}\, N^{-1}(x)$

$d_1 = 0$ for $N(d_1) = 0.50$ → $\log K = \log F + \frac{1}{2}\sigma^2(T-t)$, which is a little more than the FX forward rate

Using $\log F = \log S + (r_d - r_f)(T-t)$

→ **Go to python examples**

concurrent.futures -- Launching parallel tasks in python

The concurrent.futures module in python can be used to do parallel processing with python functions, across multiple threads or multiple processors. There is no need to call C++/C functions. The concurrent.futures module does not run functions on GPU's.

Documentation is available at https://docs.python.org/3/library/concurrent.futures.html

This module is useful for speeding up python code.

→ Go to the concurrent.futures example from documentation

Additional Tools for High Performance Computing with Python

1) **RAPIDS:** https://rapids.ai/

   RAPIDS is currently available on Linux operating systems only

2) **Pytorch:** https://pytorch.org/

3) **Numba:** http://numba.pydata.org/

BFGS Algorithm for Nonlinear Minimization

The Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm is a well- known algorithm for nonlinear minimization. This algorithm can be used to do model parameter calibration by using it to minimize the sum of squared errors for a model fit.

The algorithm requires a function to compute the function value and the 1st partial derivatives with respect to the parameters. One can also apply constraints on the parameter search.

We want to minimize a function, $f(x)$.
Define the gradient to be $g_k = \frac{\partial f}{\partial x}$, and let $p_k = -B_k^{-1} g_k$ be the change in parameters at step k, $x_{k+1} = x_k + \alpha_k p_k$.
If we replace B_k with the Hessian, the matrix of 2nd derivatives, we have the Newton method. The BFGS algorithm uses an approximation to the Hessian inverse calculated as follows.

Set $s_k = \alpha_k p_k$ and $y_k = g_{k+1} - g_k$.

$$B_{k+1} = B_{k+1} + \frac{y_k y_k'}{y_k' s_k} - \frac{B_k s_k s_k' B_k'}{s_k' B_k s_k}$$

$$B_k^{-1} = \left(I - \frac{s_k y_k'}{y_k' s_k}\right) B_k^{-1} \left(I - \frac{y_k s_k'}{y_k' s_k}\right) + \frac{s_k s_k'}{y_k' s_k}$$

Set the initial $B_k = I$, or an approximation. One can impose constraints on the vector of paramteres, x_k

BFGS Algorithm for Nonlinear Minimization

See C++ function for Constrained BFGS in class example code

BFGS is a popular algorithm for doing parameter search and parameter calibration, and it can be used to do parameter calibration in financial models.