Big Data in GPGPUs

May 29, 2019

UT Austin Visualization group

Immersive Training in Advanced Scientific Visualization
June 11-13, 2019

Are you ready to get more from your data than simple charts?

Spend three days with Texas Advanced Computing Center's visualization experts and learn how to generate rich, informative visualizations for your simulation results and data analyses.

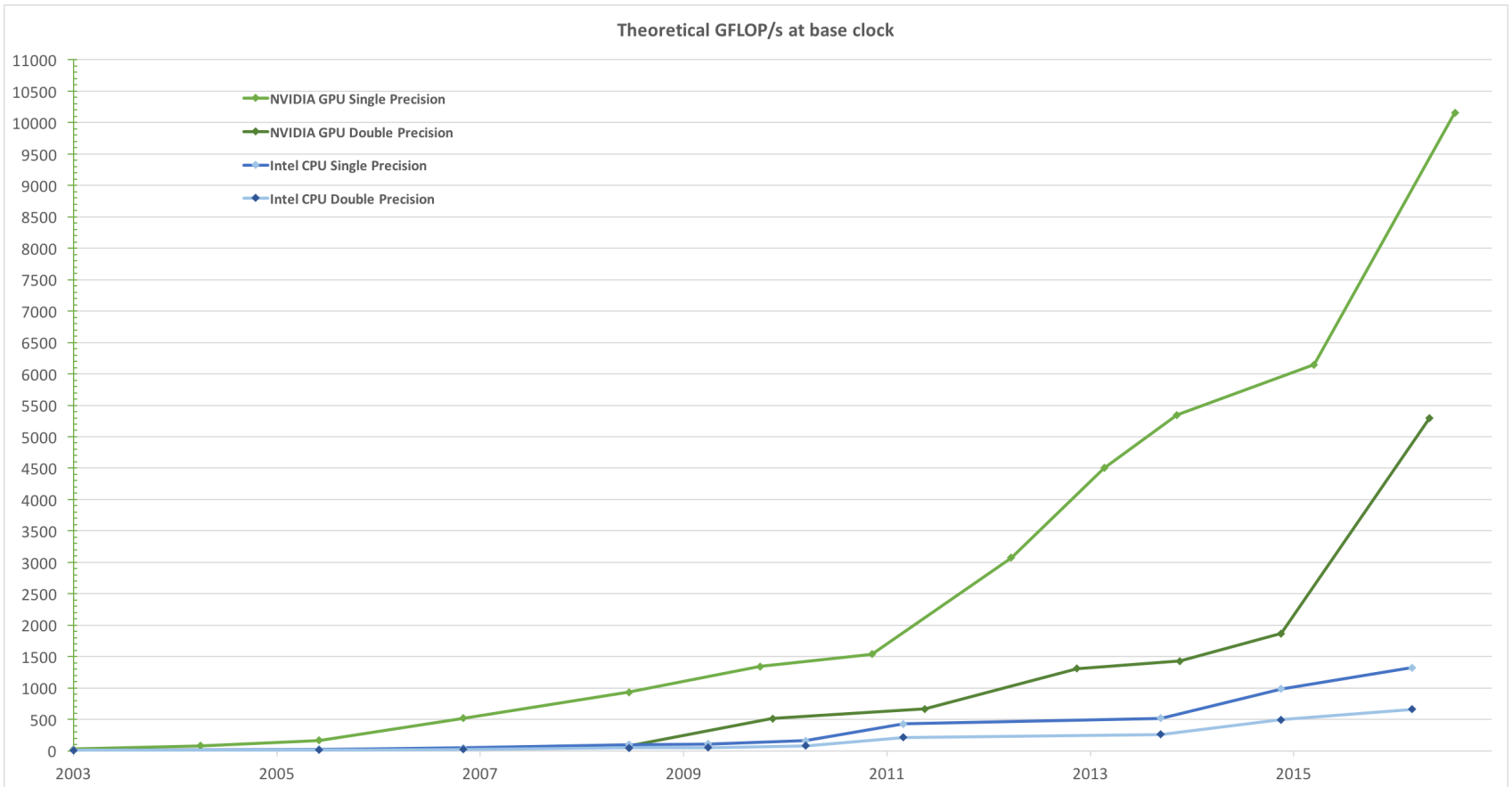
Get started with VisIt and Paraview, learn the basics of developing interesting and worthwhile geographic and information visualizations.

https://utaustinportugal.org/events/immersive-training-in-advanced-scientific-visualization/

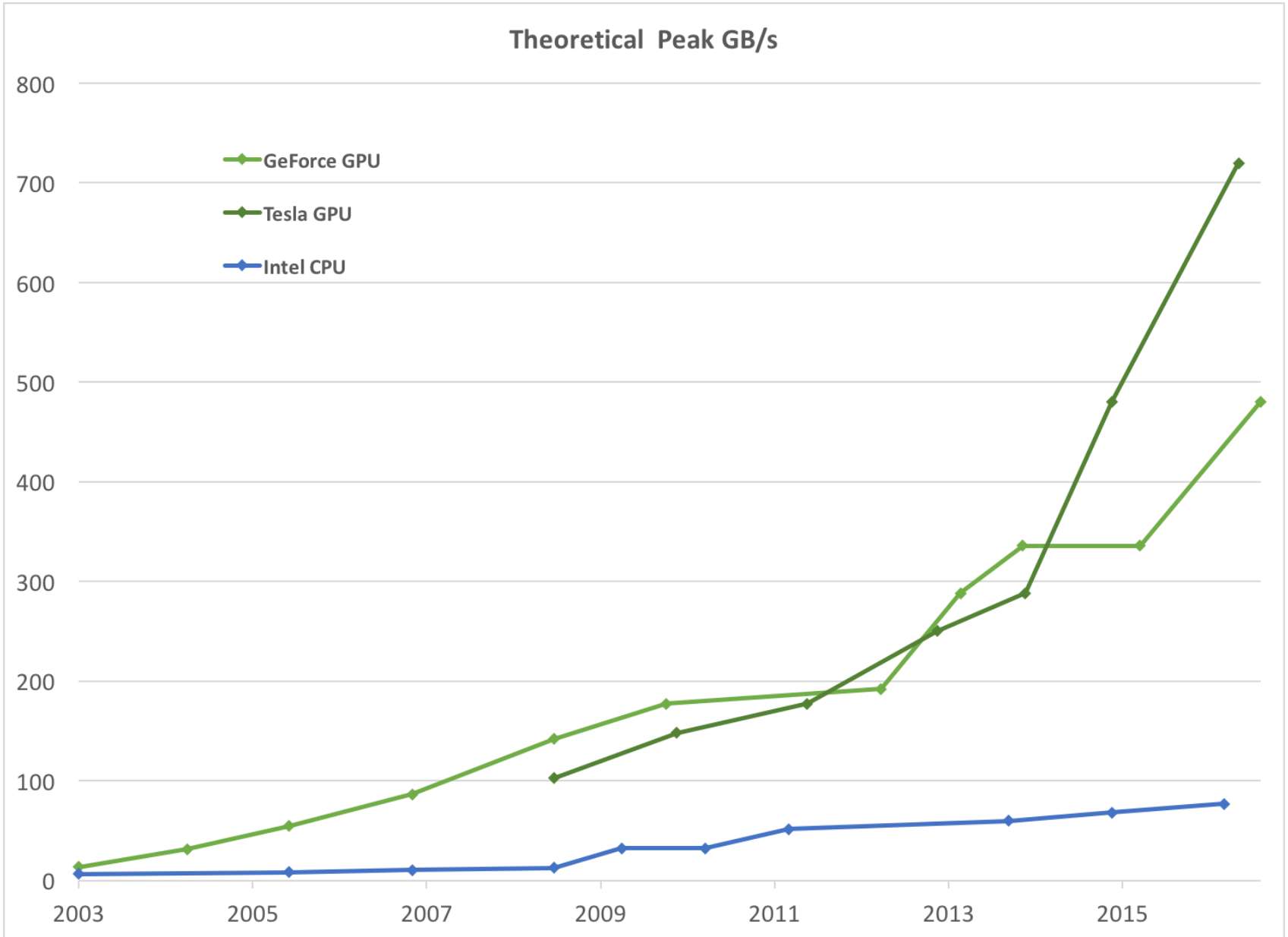
Big Data on GPGPUs

- General Purpose Graphical Processing Units (GPGPUs) focus on data-parallel computations rather than task-parallelism
- Scalable array of multithreaded Streaming Multiprocessors (SMs)

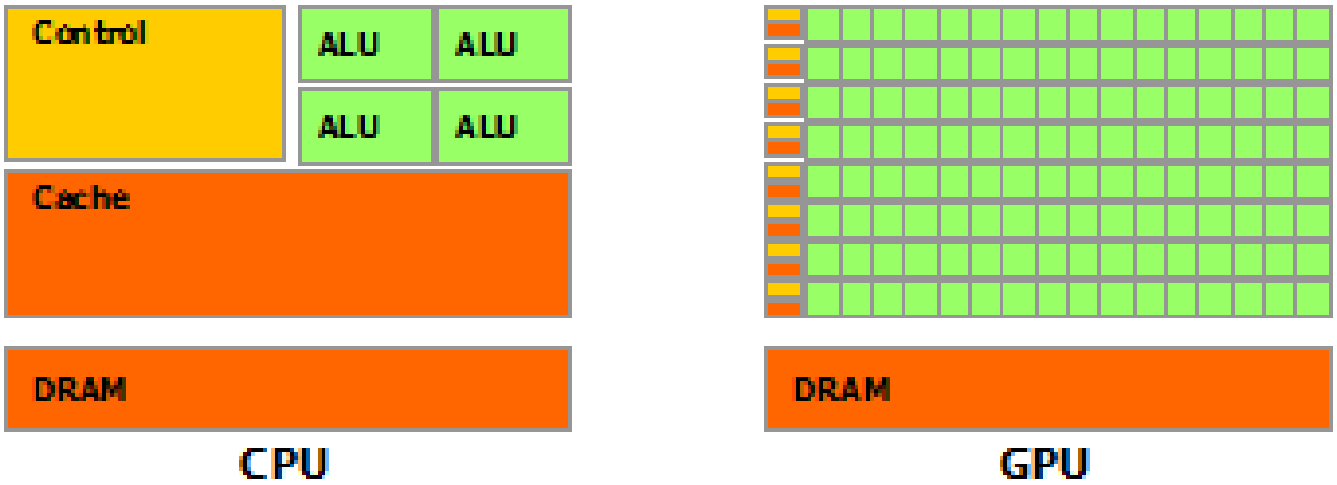
GPU Architecture



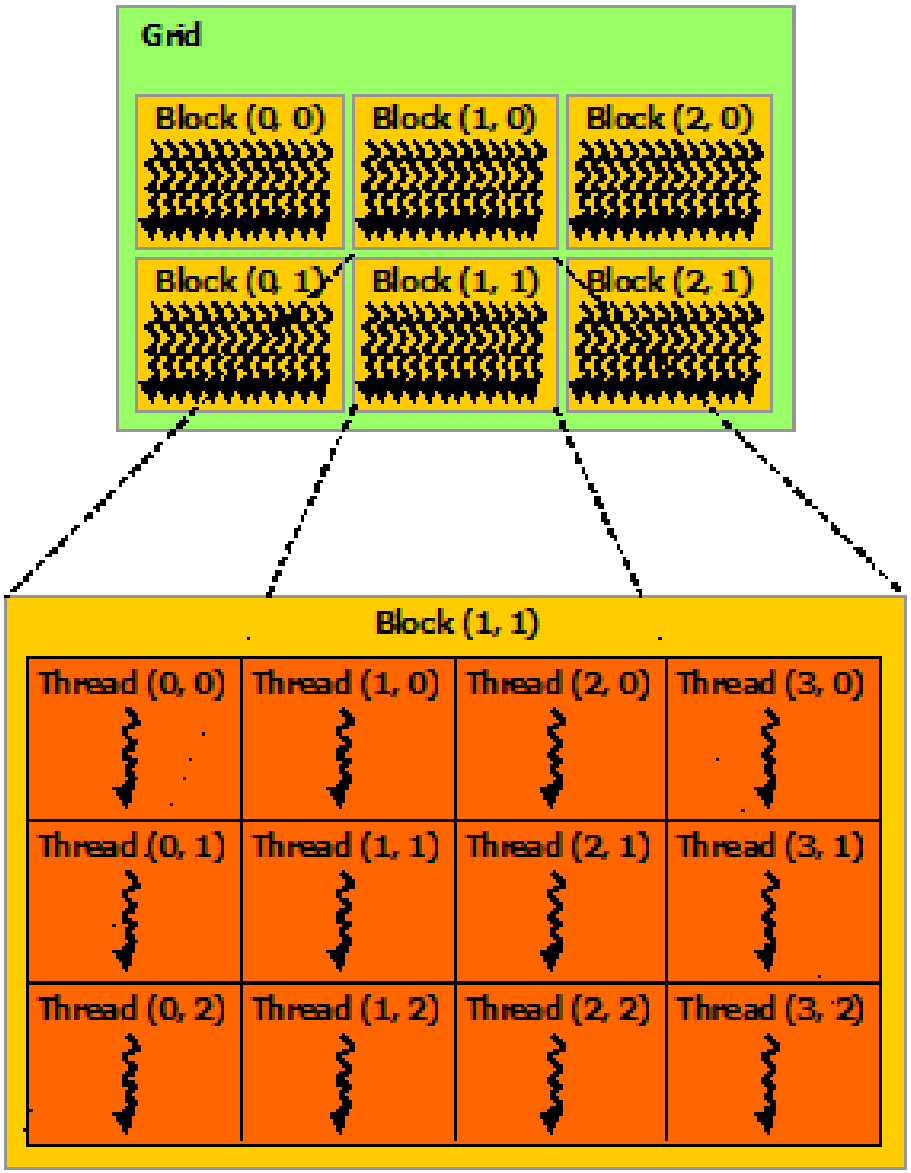
GPU Architecture



GPU Architecture



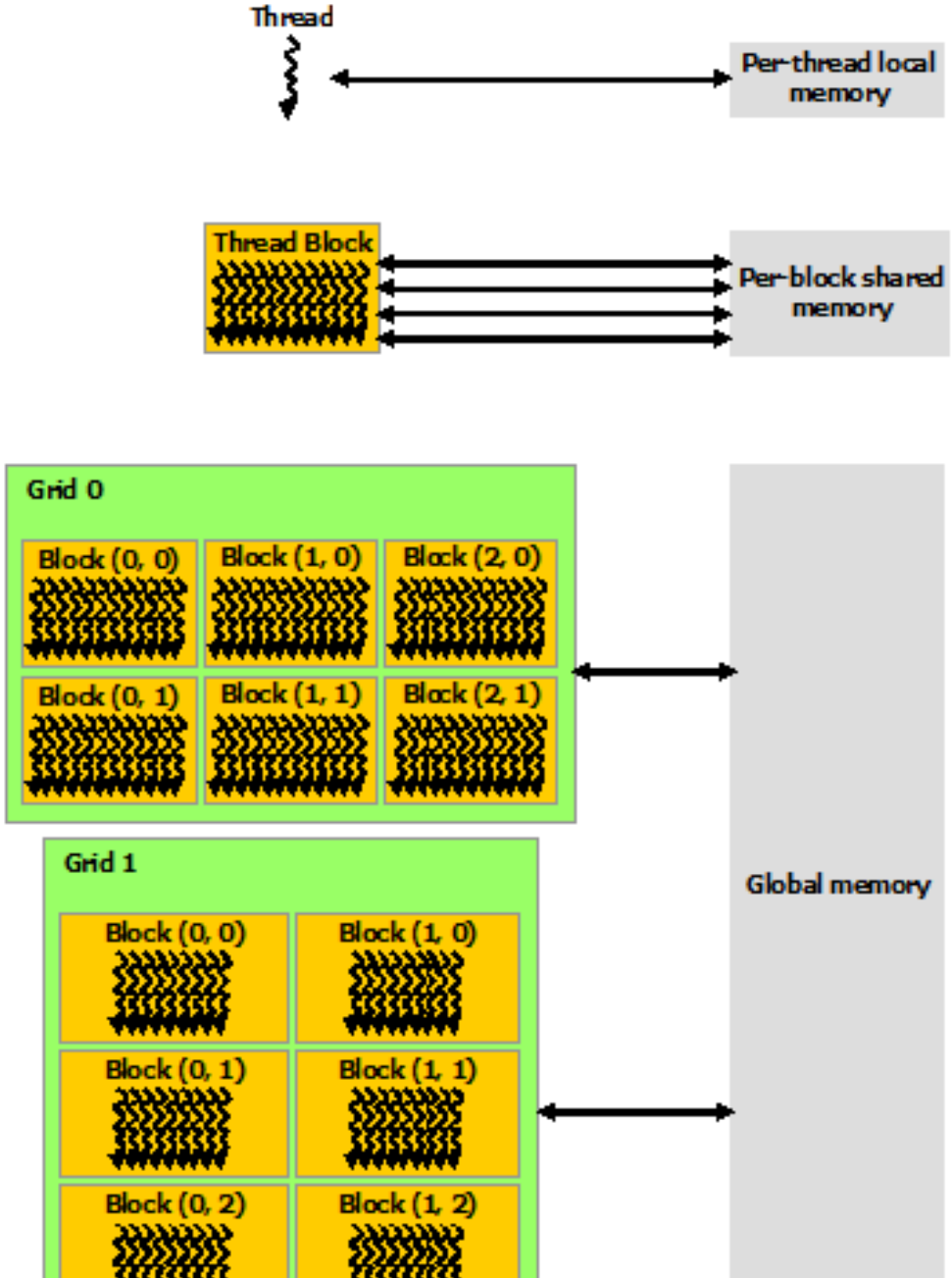
GPU Architecture



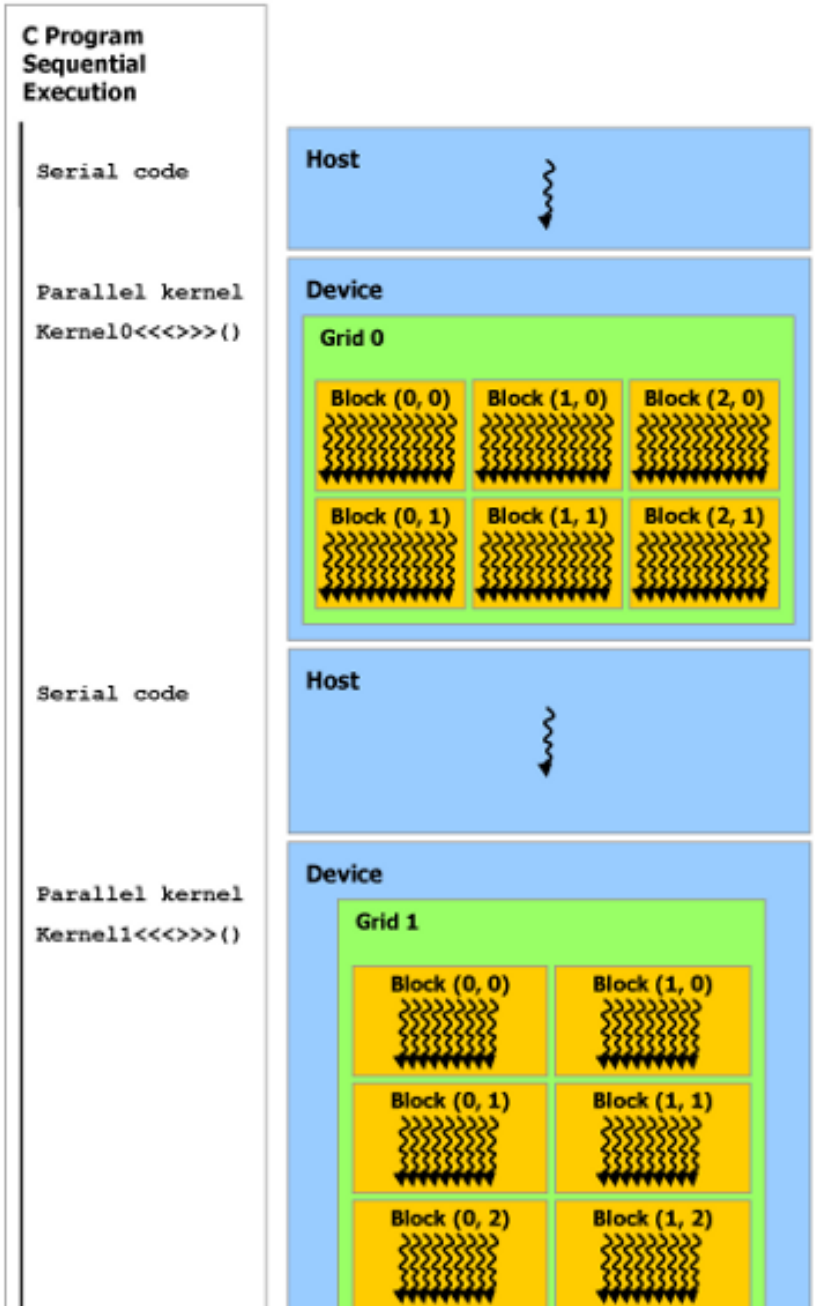
Grids, blocks and threads

- Usually, a grid is organized as a 2D array of blocks
- A block is organized as a 3D array of threads
- Both grids and blocks use the dim3 type with three unsigned integer fields
- Unused fields are initialized to 1 and ignored.

GPU Architecture



Heterogeneous programming



Data Partitioning

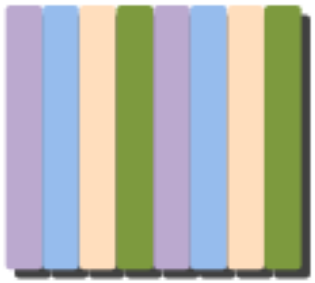
Block partition: each thread takes one data block

Cyclic partition: each thread takes two data blocks

FIGURE 1-4

Block partition on one dimension

Block partition on both dimensions



Cyclic partition on one dimension

(from http://www.hds.bme.hu/~fhegedus/C++/Professional%20CUDA%20C%20Programming.pdf)

Alternatives for python

- PyCUDA or PyOpenCL
  (slides from https://www.pycon.it/media/conference/slides/gpu-accelerated-data-analysis-in-python-a-study-case-in-material-s pdf)
- Numba
  (slides from https://devblogs.nvidia.com/numba-python-cuda-acceleration/)

PyCUDA: workflow

PyCUDA Workflow: "Edit-Run-Repeat"

A two-fold aim:

1. usage of *existing* `CUDA C`
2. *on top* of the first layer, PyCUDA $\Rightarrow$ *abstractions*

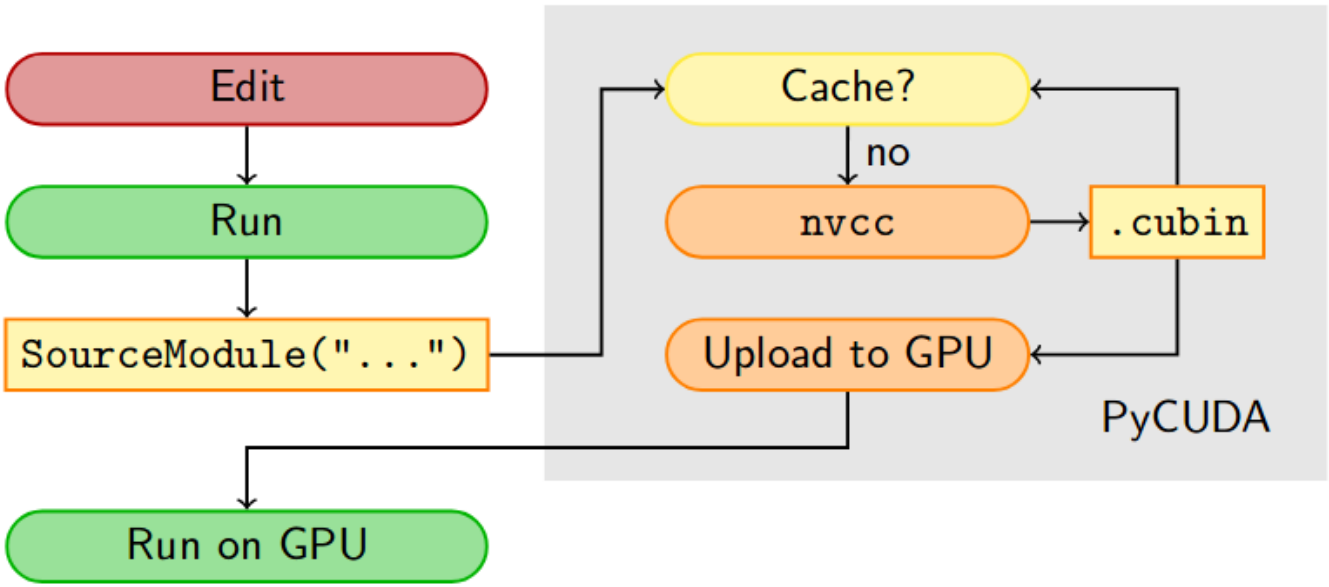


Figure: A. Klöckner et al. 2013, https://arxiv.org/abs/1304.5553

PyCUDA: gpuarrays

Using abstraction: GPUArrays

```
import numpy as np
import pycuda.autoinit
import pycuda.gpuarray as gpuarray

a_gpu = gpuarray.to_gpu(np.random.randn(4,4).astype(np.float32))
a_doubled = (2*a_gpu).get()
print(a_doubled)
print(a_gpu)
```

GPUArrays: computational linear algebra

- element-wise algebraic operations (+, -, *, /, sin, cos, exp)
- tight integration with numpy
  - gpuarray.to_gpu(numpy_array)
  - numpy_array = gpuarray.get()
- mixed data types (int32 + float32 = float64)

PyCUDA: hello world! (1)

The "Hello World" of PyCUDA: the Kernel, Part I

```
import numpy as np
import pycuda.driver as drv # import PyCUDA
import pycuda.autoinit # initialize PyCUDA
from pycuda.compiler import SourceModule

mod = SourceModule("""
  __global__ void add_them(float *dest, float *a, float *b)
{
int idx = threadIdx.x; // unique thread ID within a block
dest[idx] = a[idx] + b[idx];
}
""")
```

Compute Kernel

```
add_them = mod.get_function("add_them")
a = np.random.randn(400).astype(np.float32)
b = np.random.randn(400).astype(np.float32)
dest = np.zeros_like(a) # automatic allocated space on device
add_them(drv.Out(dest), # immediate invocation style
         drv.In(a), drv.In(b),
         block=(400,1,1), grid=(1,1)) # explicit memory copies
print(dest - a+b)
```

PyCUDA: hello world! (2)

The "Hello World" of PyCUDA: the Kernel, Part II

```
import numpy as np
import pycuda.driver as drv # import PyCUDA
import pycuda.autoinit # initialize PyCUDA
from pycuda.compiler import SourceModule

a = np.random.randn(4,4).astype(np.float32) # host memory
a_gpu = drv.mem_alloc(a.nbytes) # allocate device memory
drv.memcpy_htod(a_gpu, a) # host-to-device

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

func = mod.get_function("multiply_by_two")
func(a_gpu, block=(4,4,1)) # launching the kernel
a_doubled = np.empty_like(a)
drv.memcpy_dtoh(a_doubled, a_gpu) # fetching the data
print(a_doubled)
```

Compute Kernel

PyCUDA: device properties

How to Query Device Properties

Querying Device Properties with PyCUDA

```
import pycuda.driver as drv
import pycuda.autoinit
print("PyCUDA version:pycuda.VERSION_TEXT)
print("%d device(s) found." % drv.Device.count())
for ordinal in range(drv.Device.count()):
  dev = drv.Device(ordinal)
  print("Device Number:  %d Device Name:  %s" % (ordinal, dev.name()))
  print(" Compute Capability:  %d.%d" % dev.compute_capability())
  print(" Max Thread per Block:  %d" % dev.max_threads_per_block)
  print(" Max Block dim Z: %d" % dev.max_block_dim_z)
  print(" Total Memory:  %s KB" % (dev.total_memory()//(1024)))
  print(" Memory Clock Rate (KHz):  %d" % dev.clock_rate)
  print(" Memory Bus Width (bits):  %d" % dev.global_memory_bus_width)
  print(" Peak Memory Bandwidth (GB/s):  %f" %
2.0*dev.clock_rate*(dev.global_memory_bus_width/8)/1.0e6)
```

Output

```
PyCUDA version:  2017.1.1
2 device(s) found.
Device Number:  0 Device Name:  GeForce GTX 980
 Compute Capability:  5.2
 Max Thread per Block:  1024
 Max Block dim Z: 64
 Total Memory:  4135040 KB
 Memory Clock Rate (KHz):  1215500
 Memory Bus Width (bits):  256
 Peak Memory Bandwidth (GB/s):  77.792
```

Numba

- Python compiler from Anaconda
- Compile Python code for execution on CUDA-capable GPUs or multicore CPUs
- Numba team implemented pyculib that provides a Python interface to CUDA libraries:
  - cuBLAS (dense linear algebra)
  - cuFFT (Fast Fourier Transform)
  - cuRAND (random number generation)

Numba example (1)

```
import numpy as np
from numba import vectorize

@vectorize(['float32(float32, float32)'], target='cuda')
def Add(a, b):
  return a + b

# Initialize arrays
N = 100000
A = np.ones(N, dtype=np.float32)
B = np.ones(A.shape, dtype=A.dtype)
C = np.empty_like(A, dtype=A.dtype)

# Add arrays on GPU
C = Add(A, B)
```

Numba example (2)

```
import numpy as np
from pyculib import rand as curand

prng = curand.PRNG(rndtype=curand.PRNG.XORWOW)
rand = np.empty(100000)
prng.uniform(rand)
print rand[:10]
```

Numba example: Mandelbrot

https://github.com/harrism/numba_examples/blob/master/mandelbrot_numba.ipynb