N-ways to GPU Programming Bootcamp

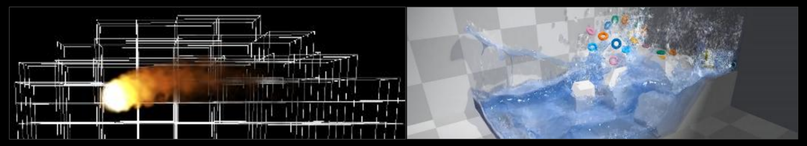
Accelerating Python on GPUs

9th April 2025

Paul Graham, Senior Solutions Architect, NVIDIA

pgraham@nvidia.com

NVIDIA



GPU Computing

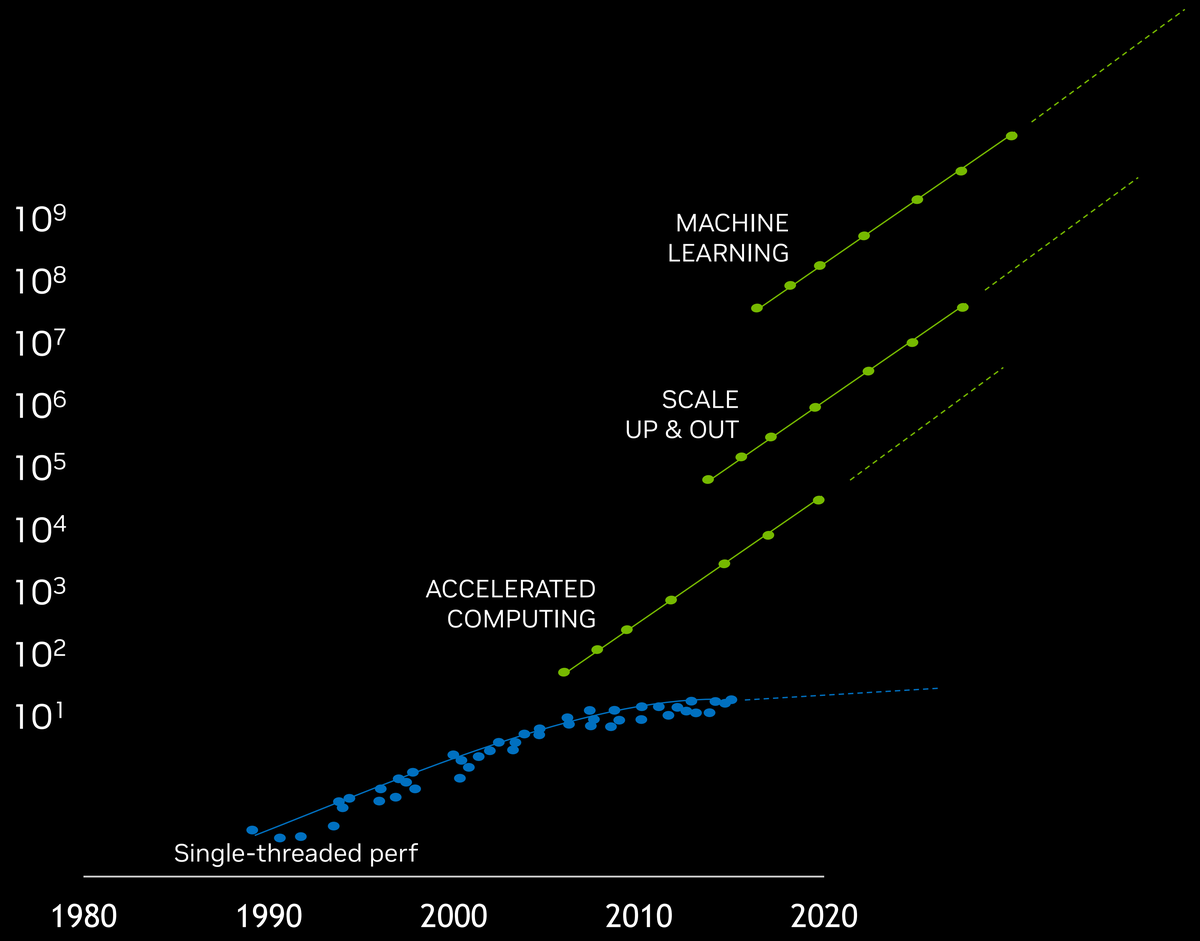
Computer Graphics

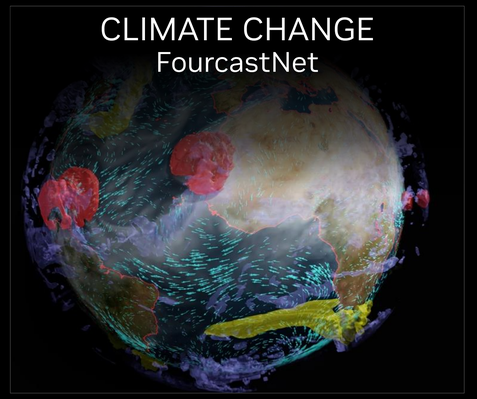
Artificial Intelligence

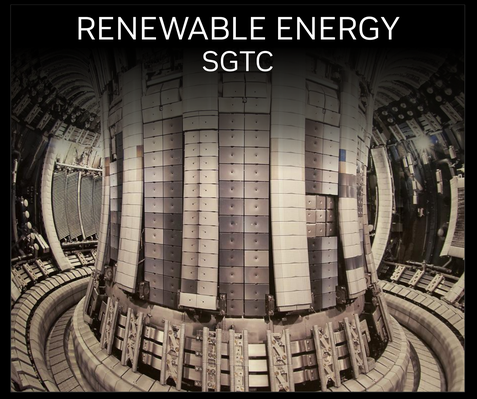
GPUs: The basics

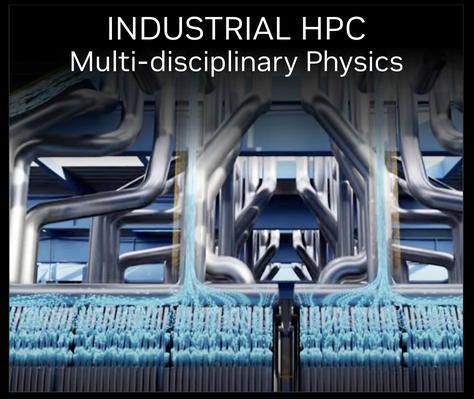
Million-X Speedup for Innovation and Discovery

Simulation + AI

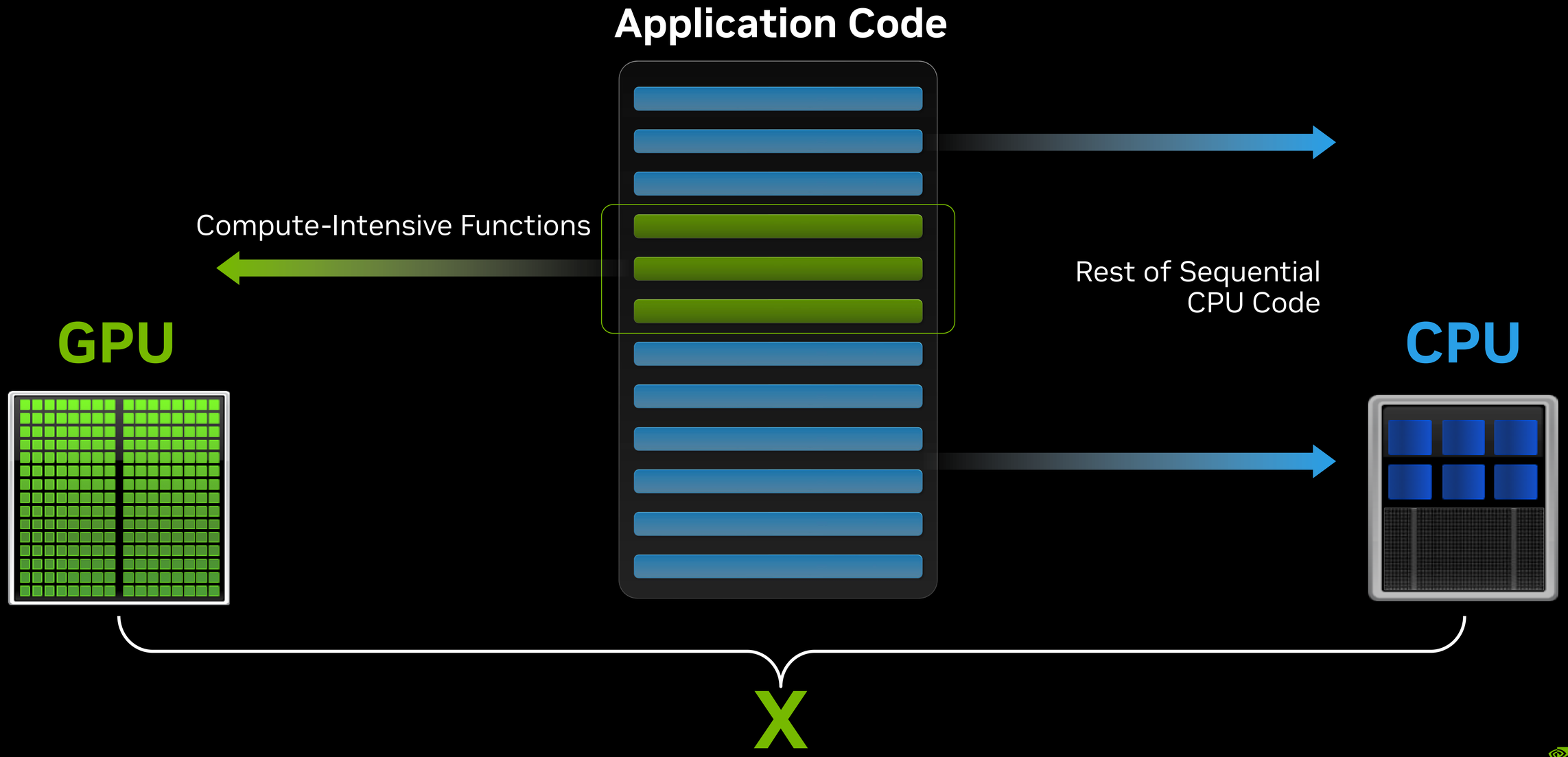








Small Changes, Big Speed-up



GH100 GPU Architecture

https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper

GH100 GPU Architecture

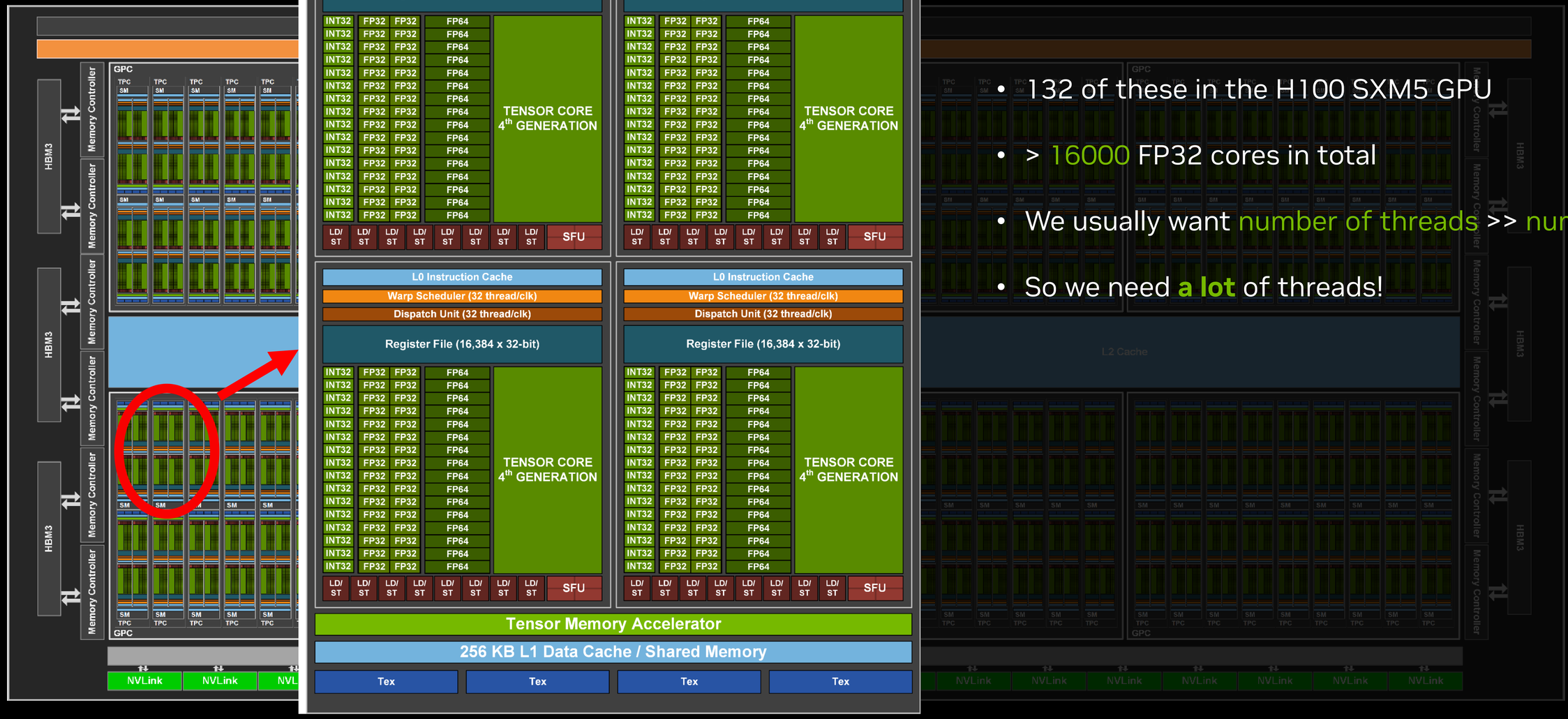
https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper

ecture

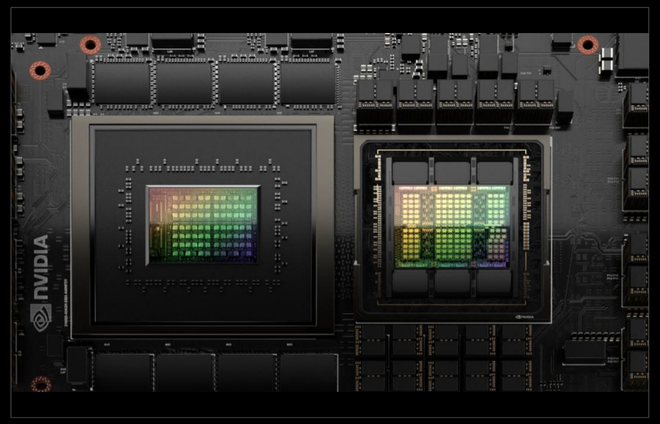
ht e/gtc22-whitepaper-hopper

ecture

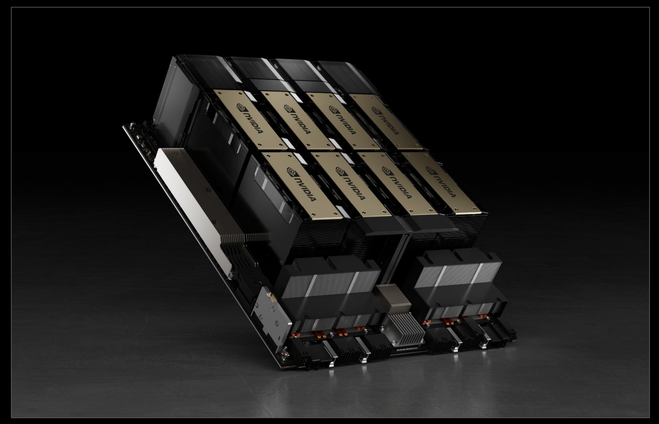
ht e/gtc22-whitepaper-hopper



- 132 of these in the H100 SXM5 GPU
- > 16000 FP32 cores in total
- We usually want number of threads >> num cores
- So we need **a lot** of threads!



Multi-die



Multi-chip

Multi-node

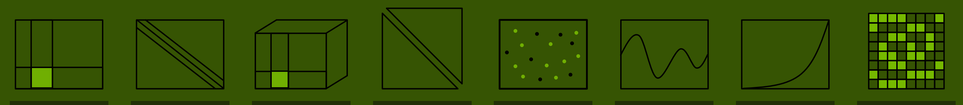
The CUDA Platform

Target the abstraction layer that works best for your application

Developer & Application Ecosystem

- Frameworks: PyTorch, TensorFlow, Jax, Modulus, Triton, ...
- SDKs: Medical Devices, Energy, Autonomous Vehicles, ...

NVIDIA Accelerated Libraries



cuBLAS | cuSPARSE | cuTENSOR | cuSOLVER | cuRAND | cuFFT | Math API | NPP

Parallel Languages

OpenACC	OpenMP	Standard C++	Python	Julia	MATLAB	...
CUDA FORTRAN	CUDA C++	OpenCL	Ada	Haskell	R	...

Compilation Stack

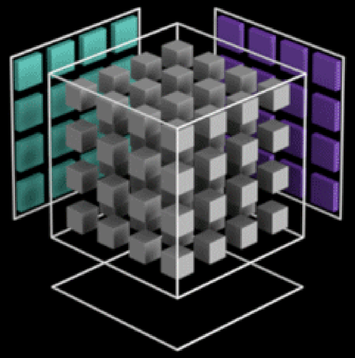
- NVVM / LLVM IR
- PTX Assembly ISA

Tensor Cores

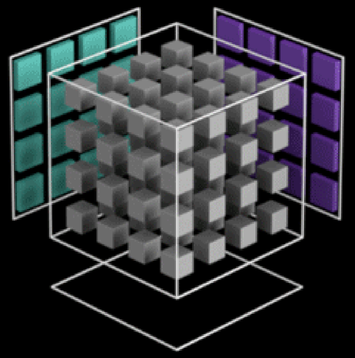
Hardware for Matrix Multiply and Accumulate operations

- Perform several MMA calcs per clock cycle
- Introduced in the V100
  - FP32 in, FP32 out (accumulate)
  - FP16 multiply
- Turing added int8, int4, int1 calculations
- Ampere (A100)
  - Full FP64 MMA
  - Bfloat16, Tensor Float 32
- Hopper (H100)
  - FP8
  - Transformer Engine

PASCAL



VOLTA TENSOR CORES



cuDNN Library, CUTLASS

Exploiting Tensor Cores

cuDNN - Accelerating deep learning primitives

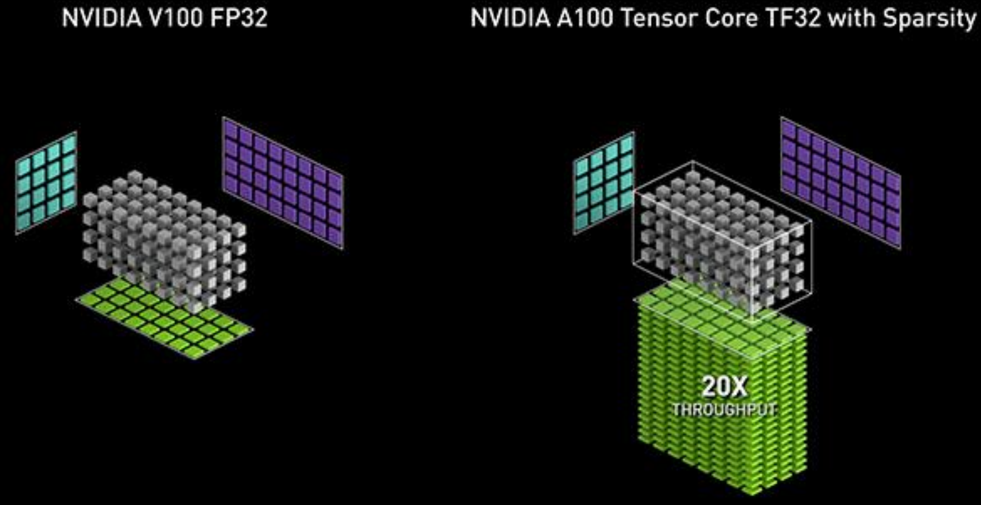
Key Features

- Tensor Core acceleration for all popular convolutions
- Supports FP32, FP16, BF16 and TF32 floating point formats and INT8, and UINT8 integer formats
- Arbitrary dimension ordering, striding, and sub-regions for 4d tensors means easy integration into any neural net implementation

Caffe · Caffe2 · Chainer · Microsoft Cognitive Toolkit · MATLAB · mxnet · PaddlePaddle · PyTorch · TensorFlow · torch · Wolfram Language

CUTLASS – Tensor Core Programming Model

- Warp-Level GEMM and Reusable Components for Linear Algebra Kernels in CUDA
- Has Python interfaces



Frameworks and Libraries

Many Frameworks … All With Python Support

NVIDIA Launchpad — free hands-on labs

- cuOpt – accelerated optimisation engine e.g. for Logistics and Route Optimisation
- Isaac Sim – robotics simulation toolkit – building virtual worlds for training robots
- Riva – speech AI services: transcription, translation, text to voice …
- Clara – AI powered solutions for healthcare and life sciences e.g. Genomics, Medical Instruments
- Holoscan – acceleration of sensor data processing pipelines

Isaac Sim

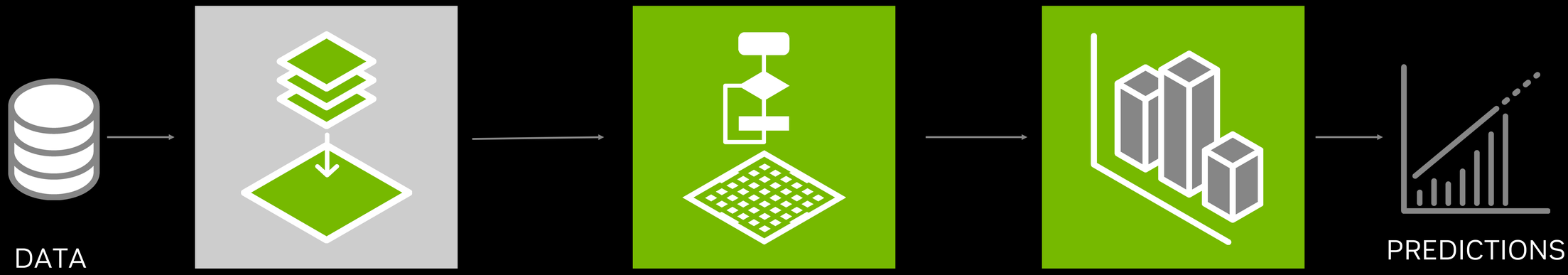
- Merlin – end to end system for recommender frameworks
- NeMo – framework for building and deploying generative AI models
- TAO toolkit – for transfer learning
- DeepStream SDK – for streaming IVA applications
- Modulus – PyTorch-based framework for Physics-informed Neural Networks (PINNs)
- . . .

NVIDIA Modulus and Omniverse

RAPIDS

GPU-accelerated data science workflow

RAPIDS.ai



DATA PREPARATION - ETL

cuDF: Python drop-in **pandas** replacement built on CUDA.

GPU-accelerated Spark

MODEL TRAINING

cuML: GPU-acceleration of popular ML algorithms e.g. **XGBoost**

Easy-to-adopt, **scikit-learn** like interface

VISUALIZATION

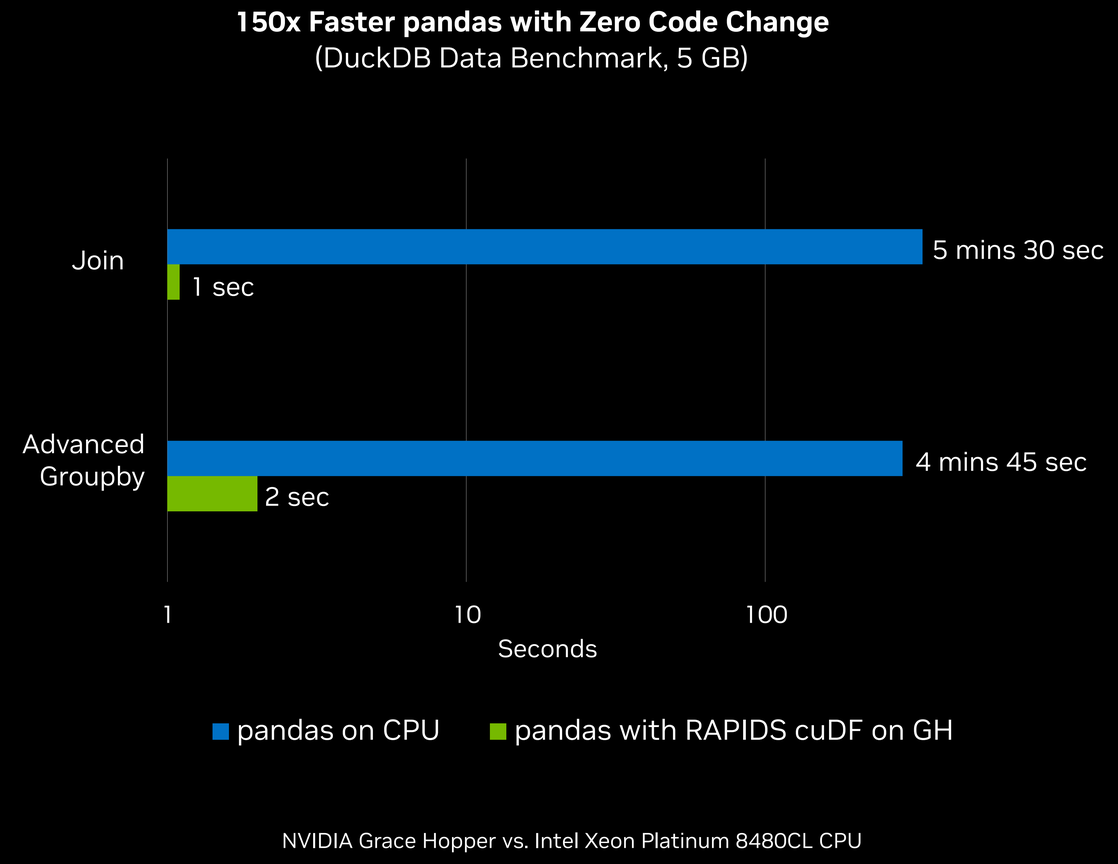
Effortless exploration of datasets, billions of records in milliseconds

Dynamic interaction with data = faster ML model development

Announcing RAPIDS cuDF Accelerates pandas with Zero Code Change

World's fastest data analytics with pandas

- 150x Faster than CPU-only
- Unified workflow on CPUs and GPUs across laptops, workstation & datacenter
- Compatible with third-party libraries built on pandas
- Available today in Open Beta and NVIDIA AI Enterprise support coming soon



NVIDIA Grace Hopper vs. Intel Xeon Platinum 8480CL CPU

pandas RAPIDS

- See also cuGraph – focussed on GPU-accelerated graph analytics including GNNs and **NetworkX:** blog
  - Has a zero code change backend for **NetworkX** , nx-cugraph

** NEW **

nvmath-python

- Bringing NVIDIA maths libraries to the Python ecosystem
  - Performance, productivity, interoperability
  - cuBLAS, cuFFT … without need for C/C++ bindings
  - Kernel fusion for efficiency
  - Kernel autotuning
  - Interoperable – e.g. can pass PyTorch data objects directly to maths libraries
  - Supports Python logging

- Demo from GTC session "Deep Dive into Math Libraries"

Polars now GPU accelerated

- **open beta**

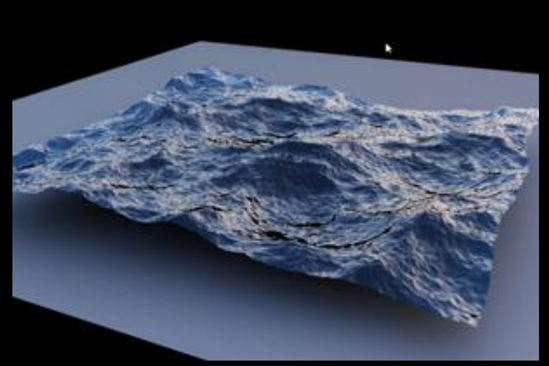
- Python DataFrame library
  - Aimed at 10s-100s GB workloads on single machine
  - Now accelerated on NVIDIA GPUs
  - Up to 13x speed up over CPUs
  - Makes use of cuDF from RAPIDS

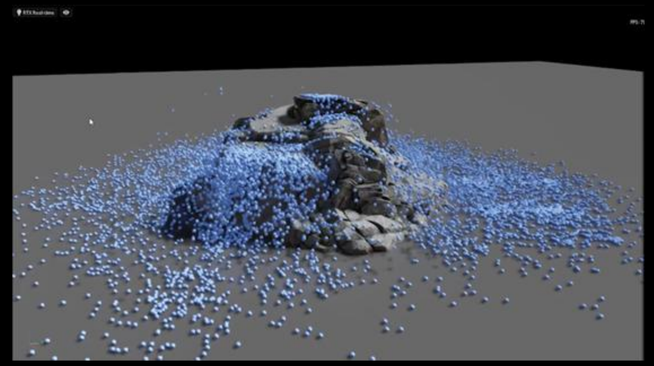
- Technical blog
- Intro notebooks: Colab | GitHub

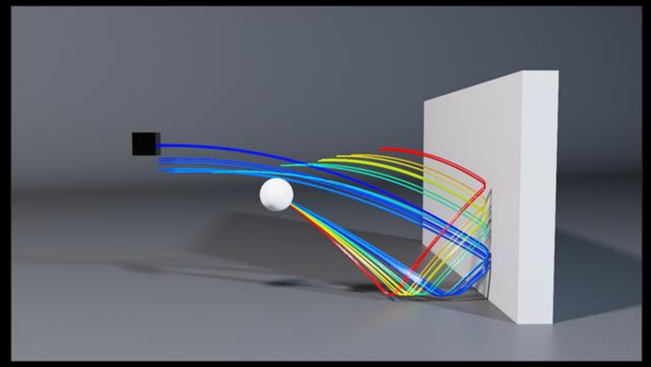
** NEW **

NVIDIA Warp

- Differentiable Spatial Computing for Python
- Developer framework for building simulation workflows
  - High-level data structures
    - Meshes
    - Sparse volumes
    - Hash grids – useful for smoothed particle hydrodynamics
  - Automatic differentiation
  - Memory model supports zero-copy data views between tensor-based frameworks
- Blog
- Tutorial: Introduction to NVIDIA Warp in Omniverse







Programming directly for GPUs

Programming the NVIDIA Platform

LANGUAGE FEATURES AND DROP-IN LIBRARIES

ISO C++, ISO Fortran, CuPy, cuPyNumeric

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y){ return y +
a*x; }
);
```

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

```
import cupynumeric as np
…
def saxpy(a, x, y):
    y[:] += a*x
```

INCREMENTAL PORTABLE OPTIMIZATION

OpenACC, OpenMP, Numba

```
#pragma acc data copy(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
...
}
```

```
@vectorize(['float64(float64, float64,
float64)'], target='cuda')
 def saxpy_ufunc(a, x, y):
    return a*x+y;
```

PLATFORM SPECIALIZATION

CUDA, Numba, PyCUDA

```
@cuda.jit(void(float32, float 32[:],
float32[:], float32[:]))
def saxpy(a, x, y, out):
    idx = cuda.grid(1)
    out[idx] = a * x[idx] + y[idx]
```

```
mod = cuda.SourceModule("""
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}
""")
```

ACCELERATION LIBRARIES

Core	Math	Communication	Data Analytics	AI	Quantum

cuPy – NumPy Compatible Library for GPU

Key Features

- Supports a subset of the `numpy.ndarray` interface
- Also makes use of NVIDIA libraries: cuBLAS, cuRAND, cuSolver …
- Can make use of Unified Memory

CPU

```
import numpy as np

def saxpy(a, x, y):
  return a * x + y

a = 3.141
x = np.random.rand(1024, 2048)
y = np.random.rand(1024, 2048)

result = saxpy(a, x, y)
```

GPU

```
import cupy as cp

def saxpy(a, x, y):
  return a * x + y

a = 3.141
x = cp.random.rand(1024, 2048)
y = cp.random.rand(1024, 2048)

result = saxpy(a, x, y)
```

cuPyNumeric – Implicitly Parallel Implementations of NumPy APIs

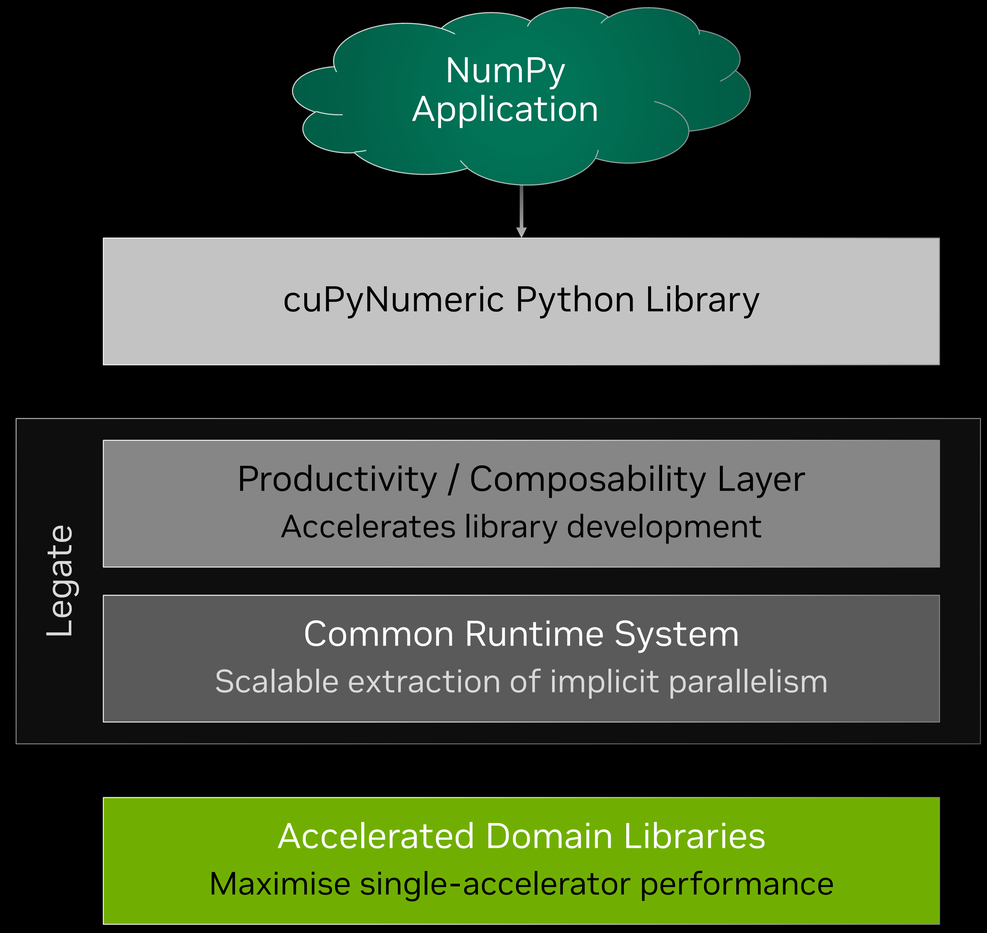
Developer blog: Effortlessly Scale NumPy from Laptops to Supercomputers with NVIDIA cuPyNumeric

Stencil Benchmark

No modifications required to scale to a thousand GPUs

numeric/blob/branch-23.03/examples/stencil.py

```
32
33  def run_stencil(N, I, warmup, timing):  # noqa: E741
34      grid = initialize(N)
35
36      print("Running Jacobi stencil...")
37      center = grid[1:-1, 1:-1]
38      north = grid[0:-2, 1:-1]
39      east = grid[1:-1, 2:]
40      west = grid[1:-1, 0:-2]
41      south = grid[2:, 1:-1]
42
43      timer.start()
44      for i in range(I + warmup):
45          if i == warmup:
46              timer.start()
47          average = center + north + east + west + south
48          work = 0.2 * average
49          center[:] = work
50      total = timer.stop()
51
52      if timing:
53          print(f"Elapsed Time: {total} ms")
54      return total
```



cuPyNumeric – Implicitly Parallel Implementations of NumPy APIs

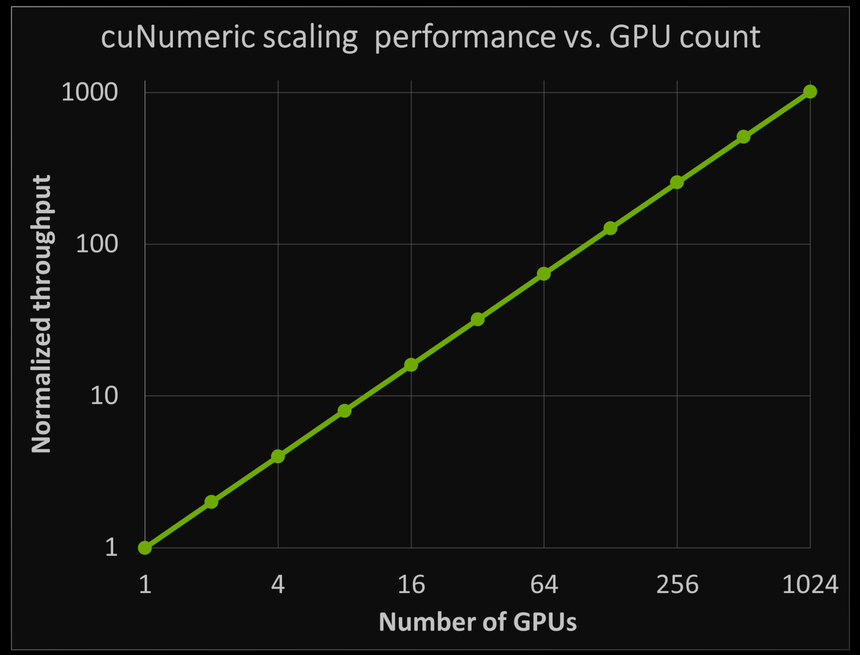
Developer blog: Effortlessly Scale NumPy from Laptops to Supercomputers with NVIDIA cuPyNumeric

Stencil Benchmark

No modifications required to scale to a thousand GPUs

numeric/blob/branch-23.03/examples/stencil.py

```
32
33  def run_stencil(N, I, warmup, timing):  # noqa: E741
34      grid = initialize(N)
35
36      print("Running Jacobi stencil...")
37      center = grid[1:-1, 1:-1]
38      north = grid[0:-2, 1:-1]
39      east = grid[1:-1, 2:]
40      west = grid[1:-1, 0:-2]
41      south = grid[2:, 1:-1]
42
43      timer.start()
44      for i in range(I + warmup):
45          if i == warmup:
46              timer.start()
47          average = center + north + east + west + south
48          work = 0.2 * average
49          center[:] = work
50      total = timer.stop()
51
52      if timing:
53          print(f"Elapsed Time: {total} ms")
54      return total
```



NVIDIA

numba – Function Annotation and/or CUDA C-like Programming

ufunc example

Key Features

- Just-In-Time (JIT) Compilation – makes use of type specialisation
- Can accelerate CPU code as well as GPU code
- Works very well with NumPy ufuncs – element-wise operations …

CPU

```
import numpy as np
from numba import vectorize

@vectorize
def saxpy(a, x, y):
  return a * x + y


a = 3.141
x = np.random.rand(1024, 2048)
y = np.random.rand(1024, 2048)

result = saxpy(a, x, y)
```

GPU

```
import numpy as np
from numba import vectorize

@vectorize([float32(float32, float32, float32)],
      target='cuda')
def saxpy(a, x, y):
  return a * x + y

a = 3.141
x = np.random.rand(1024, 2048)
y = np.random.rand(1024, 2048)

result = saxpy(a, x, y)
```

numba – Function Annotation and/or CUDA C-like Programming

kernel example

Key Features

- … also allows CUDA-style kernels for more complex algorithms

```
import numpy as np
from numba import cuda

@cuda.jit(void(float32, float 32[:], float32[:], float32[:]))
def saxpy(a, x, y, out):
  i = cuda.grid(1)  # Shorthand for cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
  out[i] = a * x[i] + y[i]

a = 3.141
x = np.random.rand(1024*2048)
y = np.random.rand(1024*2048)

d_x = cuda.to_device(x) # Make a copy of x on the GPU
d_y = cuda.to_device(y) # Make a copy of y on the GPU
d_out = cuda.device_array_like(d_y) # Create an array shaped like y on the GPU

threads_per_block = 256
blocks = 1024*2048 / threads_per_block

# Launch a GPU kernel with an appropriate execution configuration
saxpy[blocks, threads_per_block](a, d_x, d_y, d_out)
cuda.synchronize()
```

PyCUDA – Kernel Programming

Key Features

- Python interface to CUDA
- Low-level access and fine-grained control
- Can write custom kernels in C/C++ directly within Python

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

mod = SourceModule(""" # Compile the CUDA kernel code
__global__ void saxpy(int n, float a, float *x, float *y) {
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] += a*x[i];
}
""")
saxpy_cuda = mod.get_function("saxpy") # Get the function pointer for the compiled kernel

a = 3.141
x = np.random.rand(1024*2048, dtype=np.float32)
y = np.random.rand(1024*2048, dtype=np.float32)

d_x = cuda.mem_alloc(x.nbytes) # Allocate memory for x on the GPU
d_y = cuda.mem_alloc(y.nbytes) # Allocate memory for y on the GPU
cuda.memcpy_htod(d_x, x)       # Copy data from CPU to GPU
cuda.memcpy_htod(d_y, y)       # Copy data from CPU to GPU

block_dim = (256, 1, 1)
grid_dim  = ((1024*2048-1) // block_dim[0] + 1, 1)

# Launch the GPU kernel
saxpy_cuda(np.float32(a), d_x, d_y, n, block=block_dim, grid=grid_dim)

cuda.memcpy_dtoh(y, d_y) # Copy the results back to the CPU

d_x.free() # Free GPU memory
d_y.free()
```

Future plans

- Move to a more “Pythonic” way of programming:
  - cuda.core.experimental (github)
  - WeAreDevelopers article
  - 1,001 ways to write CUDA kernels in Python (GTC talk)

- CUDA: New Features and Beyond (GTC talk)

Useful Links

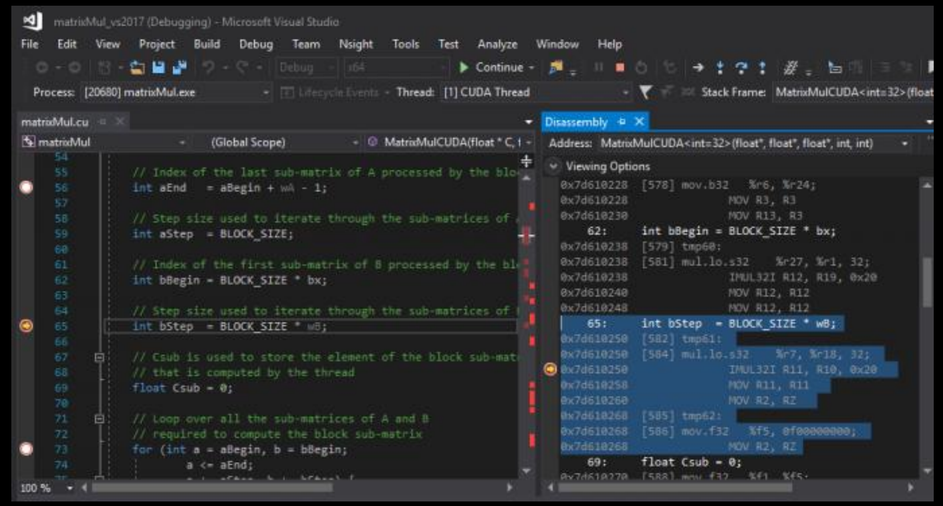
- Numba programming course
  - Fundamentals of Accelerated Computing with CUDA Python

- cuPyNumeric: https://developer.nvidia.com/cupynumeric
- Numba for CUDA GPUs: https://numba.readthedocs.io/en/stable/cuda/index.html
- CuPy: https://cupy.dev/
- PyCUDA: https://pypi.org/project/pycuda/

Resources

Developer Tools

Debuggers: cuda-gdb, Nsight Visual Studio Edition

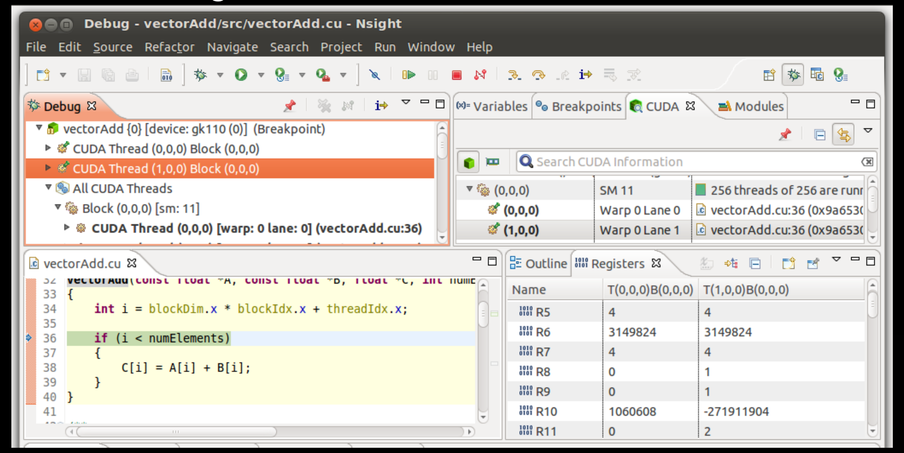


Profilers: Nsight Systems, Nsight Compute, CUPTI, NVIDIA Tools eXtension (NVTX)

Correctness Checker: Compute Sanitizer

```
$ compute-sanitizer --leak-check full memcheck_demo
========= COMPUTE-SANITIZER
Mallocing memory
Running unaligned_kernel
Ran unaligned_kernel: no error
Sync: no error
Running out_of_bounds_kernel
Ran out_of_bounds_kernel: no error
Sync: no error
========= Invalid __global__ write of size 4 bytes
=========     at 0x60 in memcheck_demo.cu:6:unaligned_kernel(void)
=========     by thread (0,0,0) in block (0,0,0)
=========     Address 0x400100001 is misaligned
```

IDE integrations: Nsight Eclipse Edition
Nsight Visual Studio Edition
Nsight Visual Studio **Code** Edition

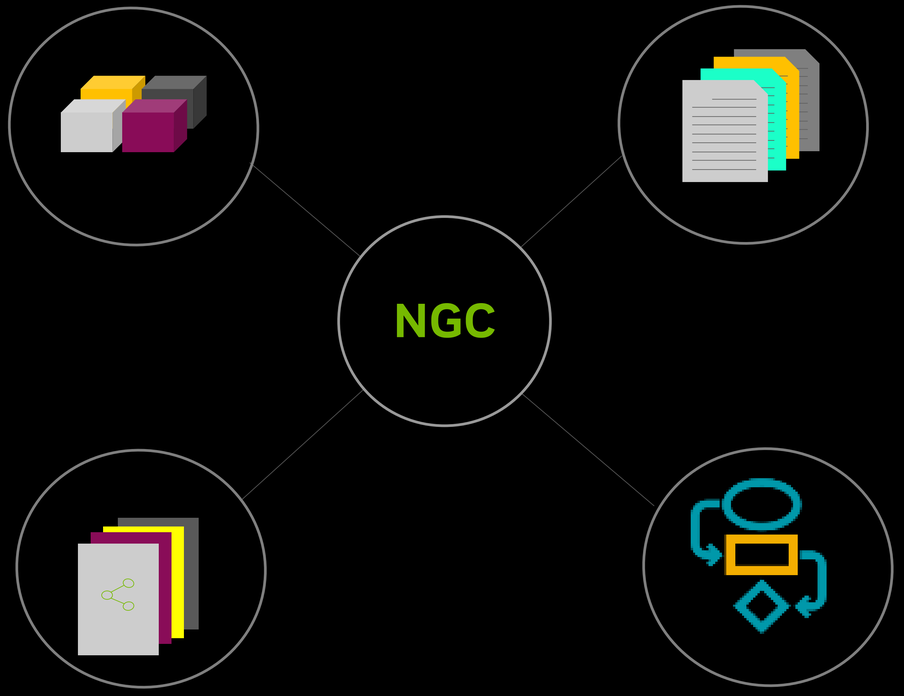


NGC: GPU-Optimized Software Hub

Simplifying DL, ML and HPC workflows

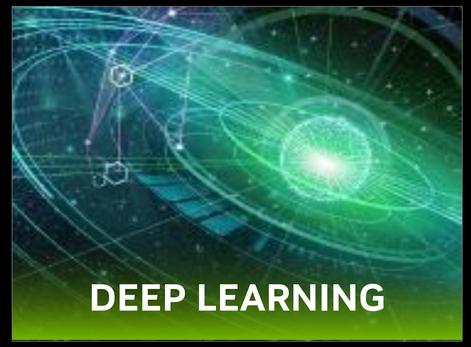
50+ Containers
DL, ML, HPC

Model Training Scripts
NLP, Image Classification, Object Detection and more



60 Pre-trained Models
NLP, Image Classification, Object Detection and more

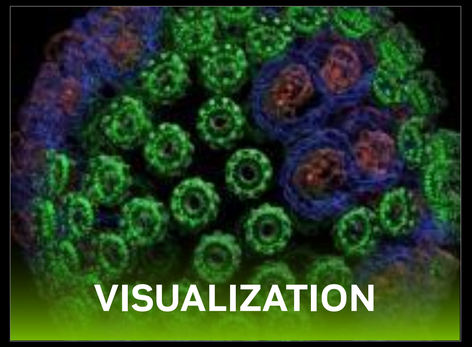
Workflows
Medical Imaging, Intelligent Video Analytics



TensorFlow | PyTorch | more

RAPIDS | H2O | more

NAMD | GROMACS | more



ParaView | IndeX | more

NVIDIA

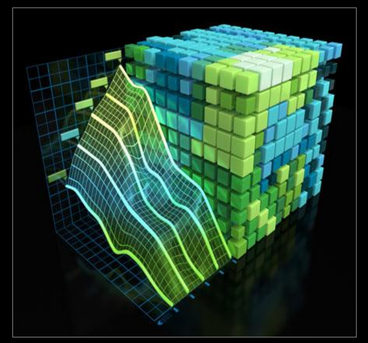
Deep Learning Institute (DLI)

Hands-on, self-paced and instructor-led training in deep learning and accelerated computing: https://www.nvidia.com/en-gb/training/

NUMBA course:
Fundamentals of Accelerated Computing with CUDA Python

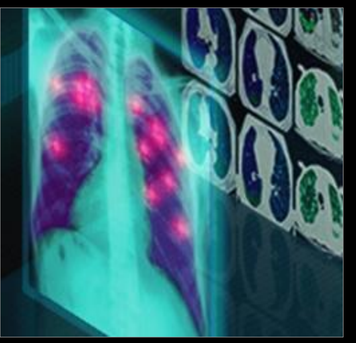
Lots of Python-based material:

- Accelerating End-to-End Data Science Workflows
- Get Started with Highly Accurate Custom ASR for Speech AI
- Introduction to Transformer-Based Natural Language Processing
- Introduction to Physics-Informed Machine Learning with Modulus
- ...



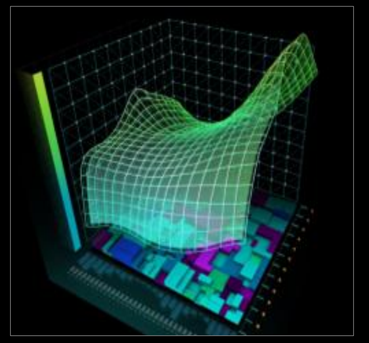
Accelerated Computing Fundamentals

Autonomous Vehicles



Medical Image Analysis

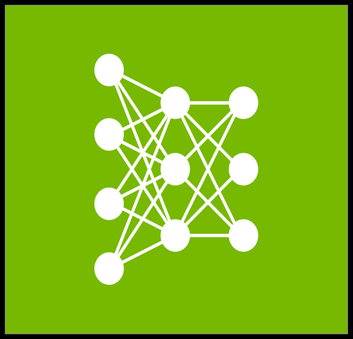
Genomics



Finance

Content

Game Development



Deep Learning Fundamentals

More industry-specific training coming soon...

Thank you!

Accelerating Python on GPUs

Paul Graham, Senior Solutions Architect

pgraham@nvidia.com