GPU BOOTCAMP

March, 2019

INTRODUCTION TO GPU COMPUTING

What to expect?

- Broad view on GPU Stack
- Fundamentals of GPU Architecture
- Ways to GPU Computing
- Good starting point

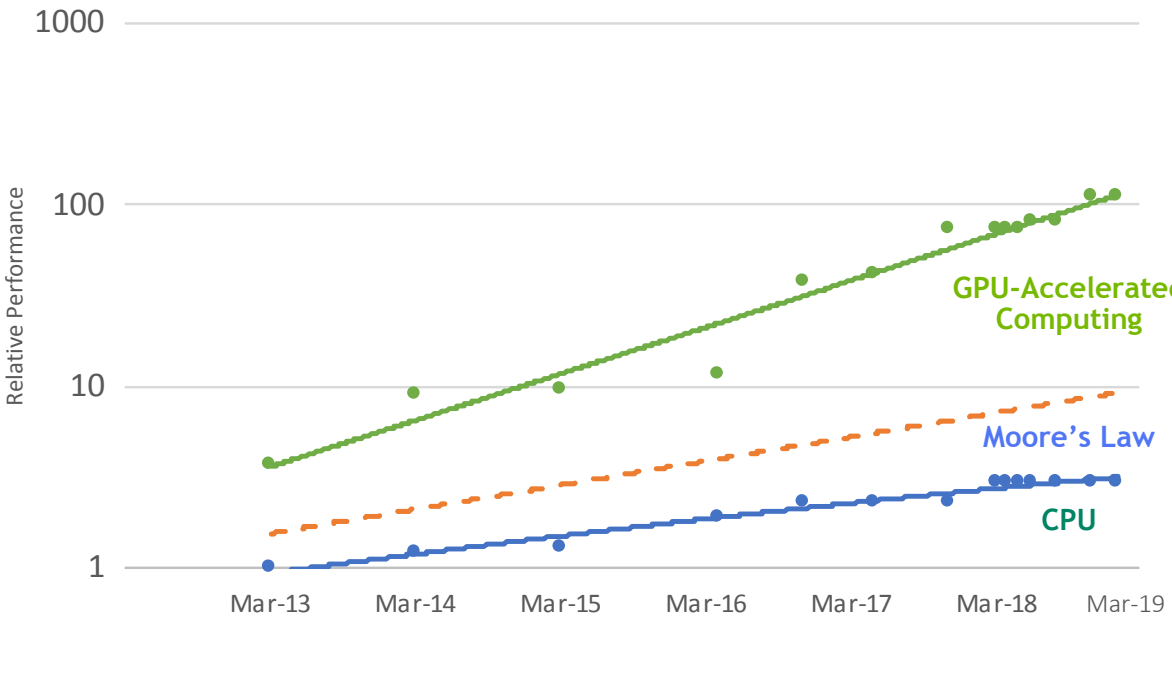
FULL STACK OPTIMIZATION

Progress Of Stack In 6 Years

2013

- cuBLAS: 5.0
- cuFFT: 5.0
- cuRAND: 5.0
- cuSPARSE: 5.0
- NPP: 5.0
- Thrust: 1.5.3
- CUDA: 5.0
- Resource Mgr: r304
- Base OS: CentOS 6.2

Accelerated Server With Fermi



Measured performance of Amber, CHROMA, GTC, LAMMPS, MILC, NAMD, Quantum Espresso, SPECFEM3D

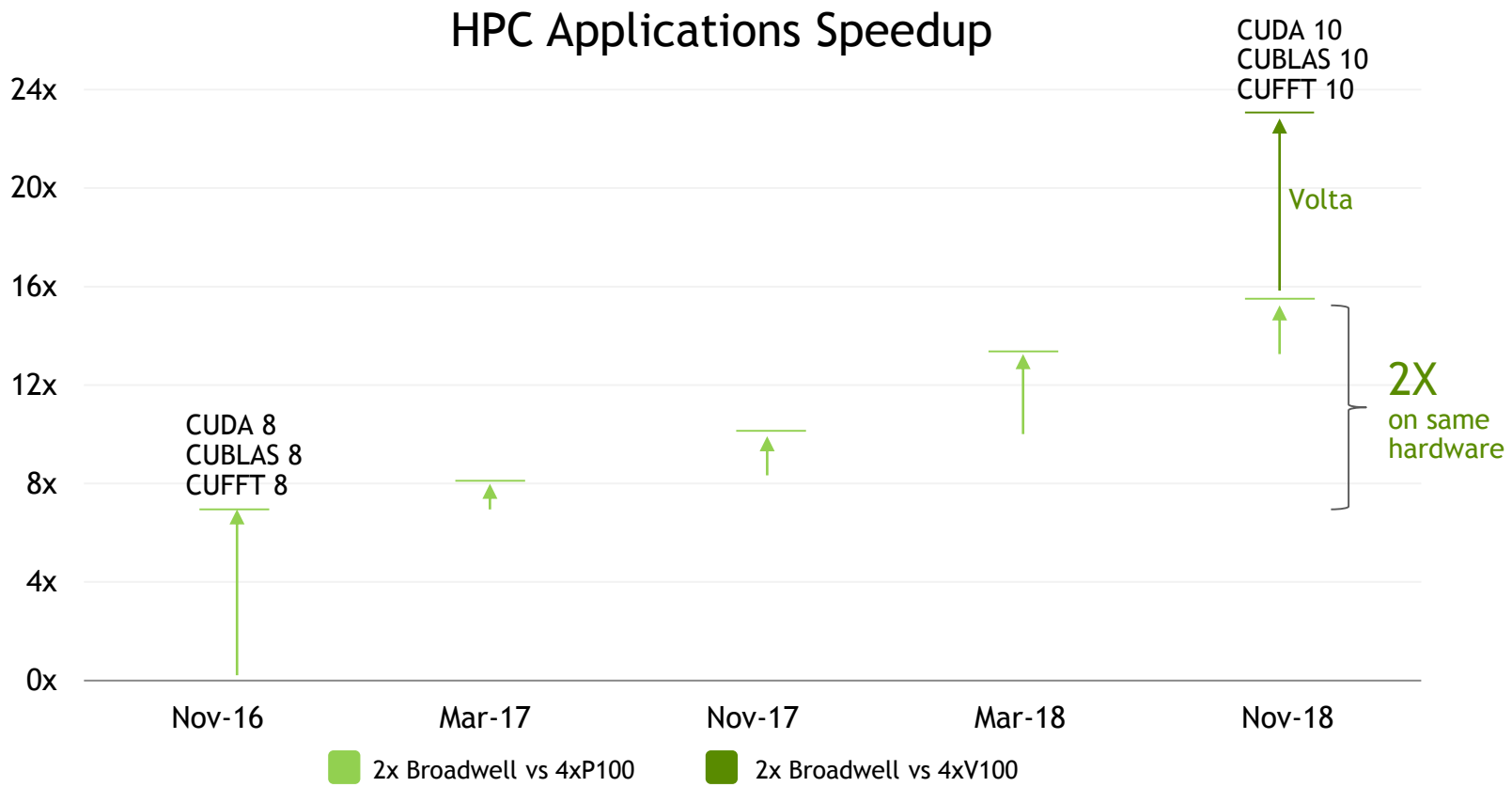
2019

- cuBLAS: 10.0
- cuFFT: 10.0
- cuRAND: 10.0
- cuSOLVER: 10.0
- cuSPARSE: 10.0
- NPP: 10.0
- Thrust: 1.9.0
- CUDA: 10.0
- Resource Mgr: r384
- Base OS: Ubuntu 16.04

Accelerated Server with Volta

ACCELERATED COMPUTING IS FULL-STACK OPTIMIZATION

2X More Performance with Software Optimizations Alone



HPC Apps: AMBER, Chroma, GROMACS, GTC, LAMMPS, MILC, NAMD, QE, RTM, SPECFEM3D, VASP

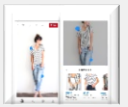
NVIDIA UNIVERSAL ACCELERATION PLATFORM

Single Platform Drives Utilization and Productivity

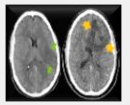
CUSTOMER USECASES


Speech

Translate


Recommender

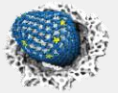
CONSUMER INTERNET

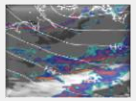

Healthcare

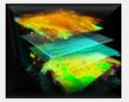

Manufacturing


Finance

INDUSTRIAL APPLICATIONS


Molecular Simulations


Weather Forecasting


Seismic Mapping

SCIENTIFIC APPLICATIONS

APPS & FRAMEWORKS

python™ RAPIDS

TensorFlow PYTORCH mxnet Chainer ONNX

Amber ANSYS NAMD SIMULIA +600 Applications

NVIDIA SDK & LIBRARIES

MACHINE LEARNING/ ANALYTICS

cuDF | cuML | cuGRAPH

DEEP LEARNING

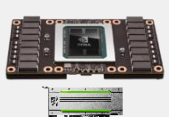
cuDNN | cuBLAS | CUTLASS | NCCL | TensorRT

HPC

CuBLAS | CuFFT | OpenACC

CUDA

TESLA GPUs & SYSTEMS


TESLA GPU


VIRTUAL GPU

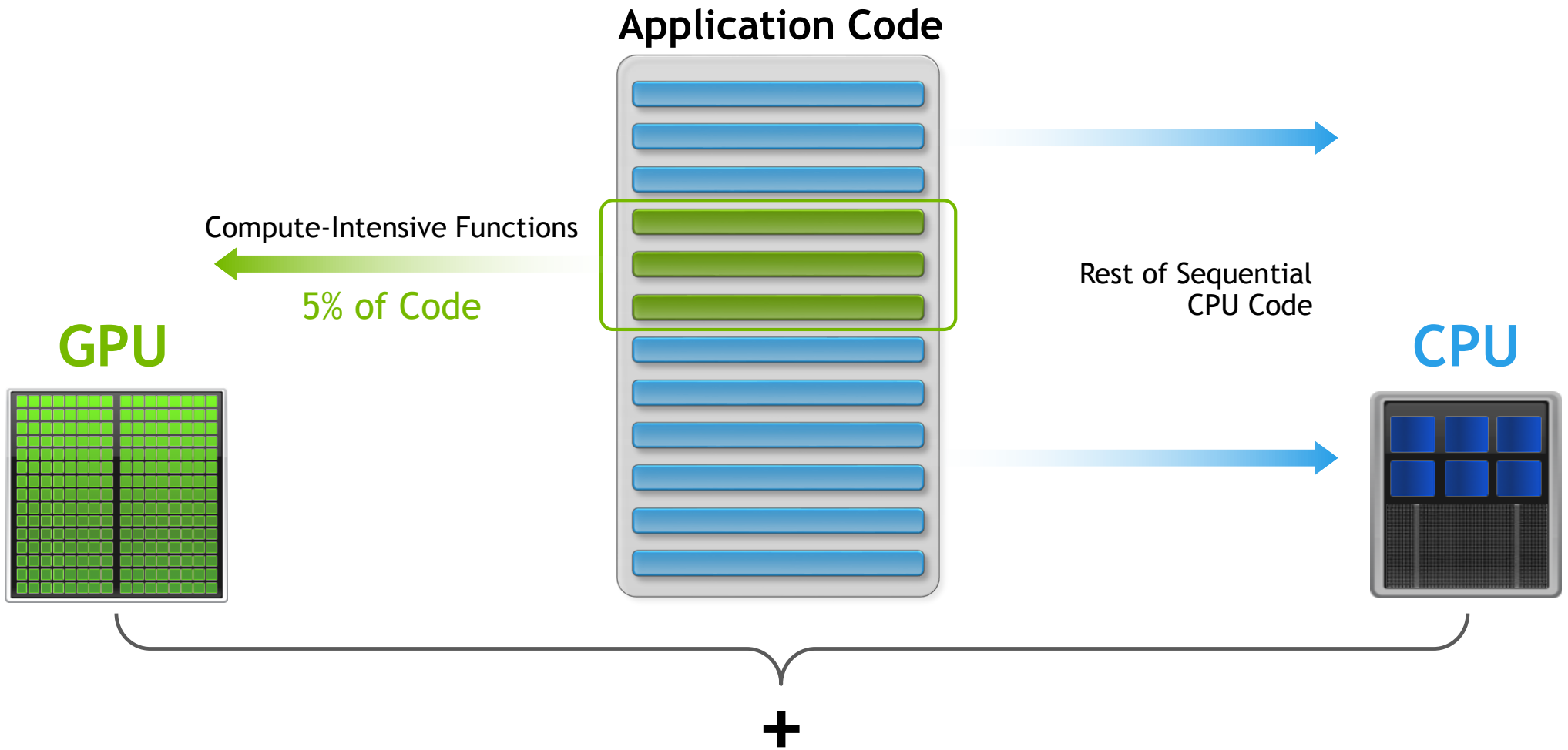

NVIDIA DGX FAMILY


NVIDIA HGX

SYSTEM OEM

CLOUD

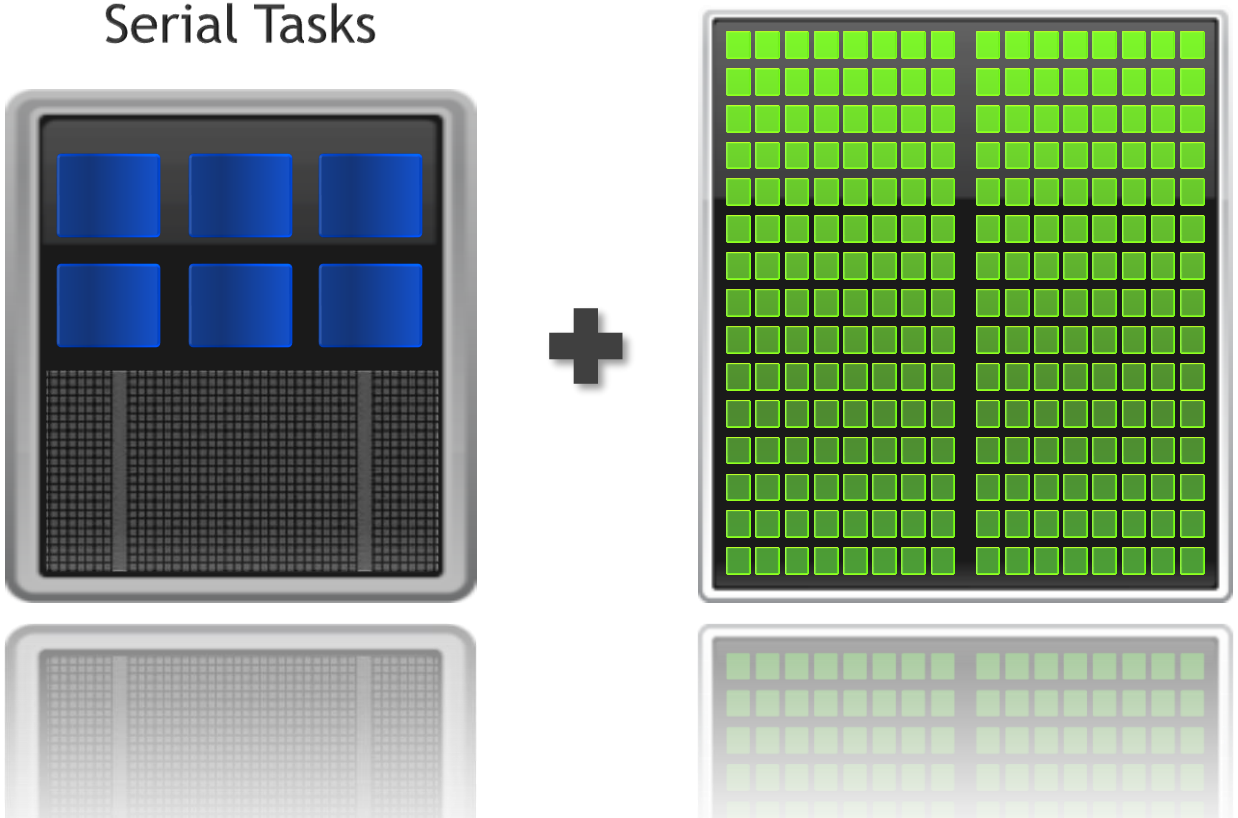
HOW GPU ACCELERATION WORKS



ACCELERATED COMPUTING

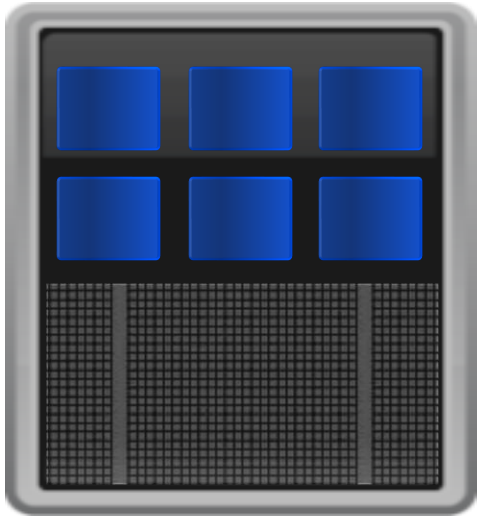
CPU
Optimized for
Serial Tasks

GPU Accelerator
Optimized for
Parallel Tasks



CPU IS A LATENCY REDUCING ARCHITECTURE

CPU
Optimized for
Serial Tasks



CPU Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- Small number of threads can run very quickly

CPU Weaknesses

- Relatively low memory bandwidth
- Cache misses very costly
- Low performance/watt

GPU IS ALL ABOUT HIDING LATENCY

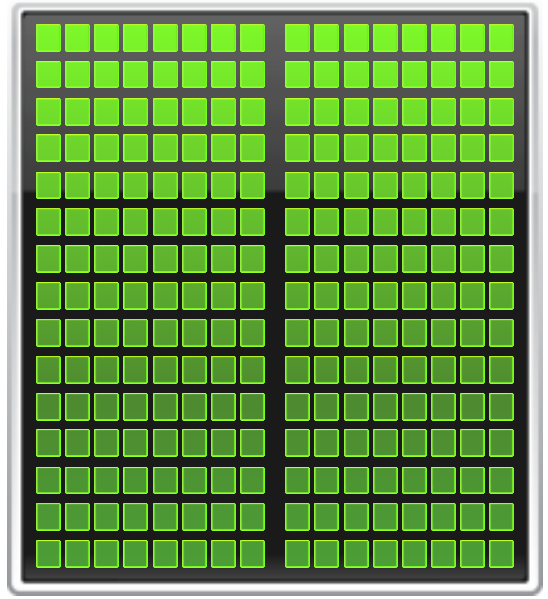
GPU Strengths

- High bandwidth main memory
- Significantly more compute resources
- Latency tolerant via parallelism
- High throughput
- High performance/watt

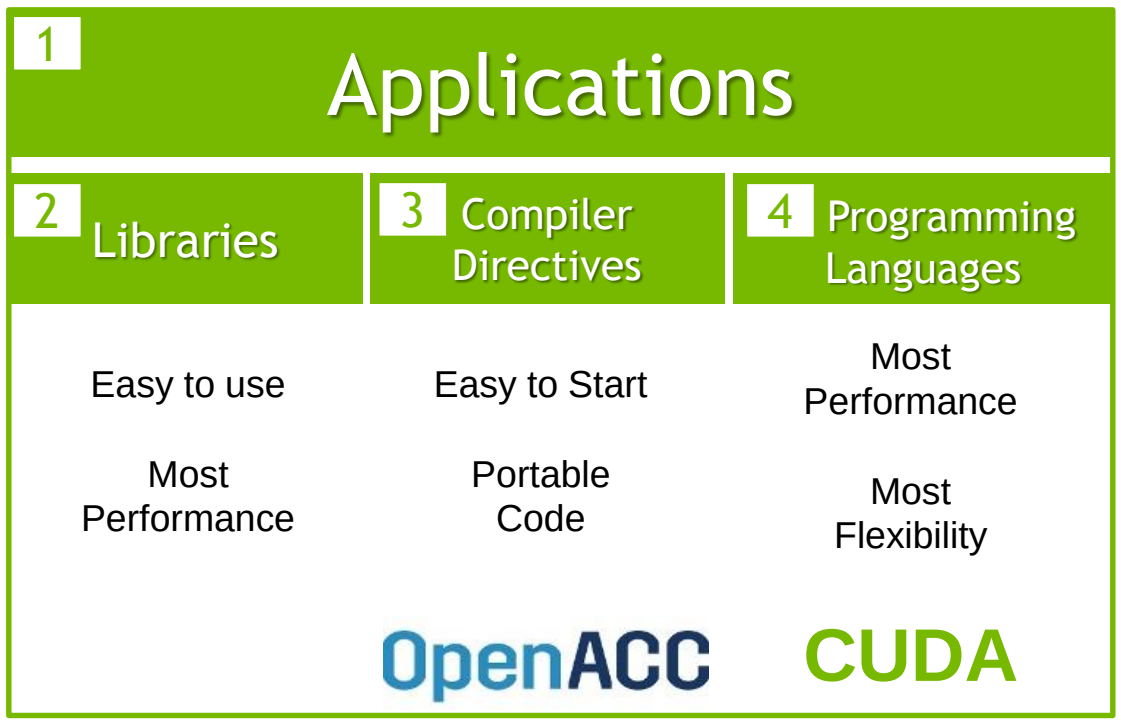
GPU Weaknesses

- Relatively low memory capacity
- Low per-thread performance

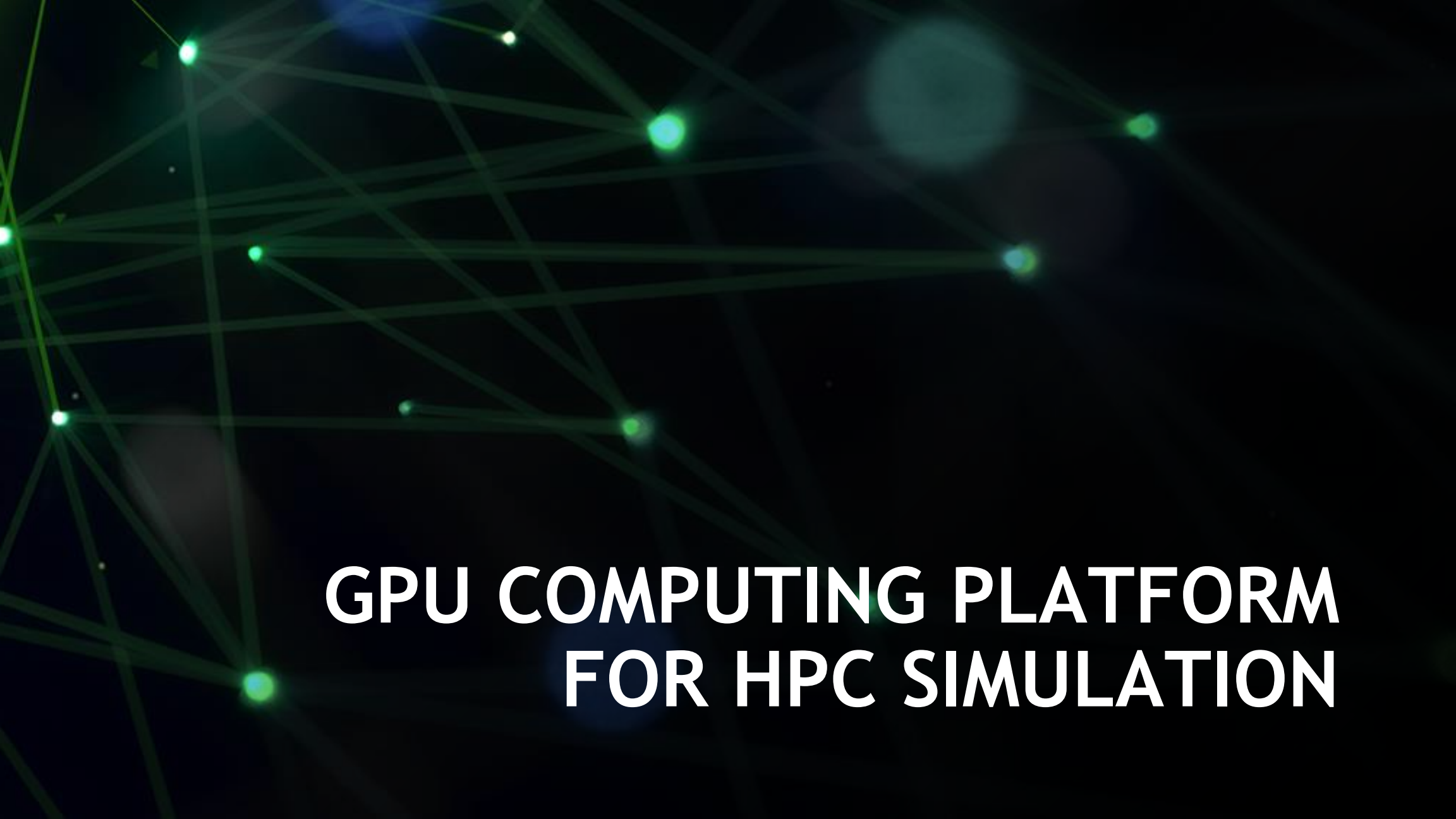
GPU Accelerator
Optimized for
Parallel Tasks



HOW TO START WITH GPUS



1. Review available GPU-accelerated applications
2. Check for GPU-Accelerated applications and libraries
3. Add OpenACC Directives for quick acceleration results and portability
4. Dive into CUDA for highest performance and flexibility


GPU COMPUTING PLATFORM
FOR HPC SIMULATION

GPU-ACCELERATED APPLICATIONS

620 Applications Across Domains

- Life Sciences
- Manufacturing
- Physics
- Oil & Gas
- Climate & Weather
- Media & Entertainment
- Deep Learning
- Federal & Defense
- Data Science & Analytics
- Safety & Security
- Computational Finance
- Tool & Management

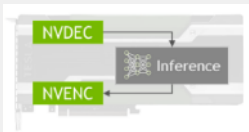
GPU ACCELERATED LIBRARIES

“Drop-in” Acceleration for Your Applications

DEEP LEARNING

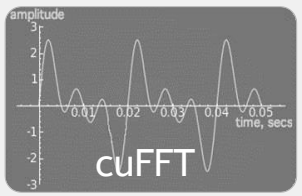
cuDNN


TensorRT



DeepStream SDK

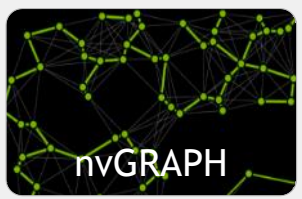
SIGNAL, IMAGE & VIDEO

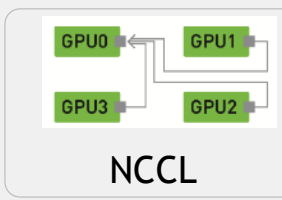


LINEAR ALGEBRA


cuSPARSE

PARALLEL ALGORITHMS





NCCL

More libraries: https://developer.nvidia.com/gpu-accelerated-libraries

WHAT IS OPENACC

Programming Model for an Easy Onramp to GPUs

Directives-based programming model for **parallel computing**

Add Simple Compiler Directive

```
main()
 {
  <serial code>
  #pragma acc kernels
  {
    <parallel code>
  }
 }
```

Designed for **performance portability** on CPUs and GPUs

Simple

Powerful & Portable

Read more at www.openacc.org/about

OpenACC is an open specification developed by OpenACC.org consortium

SINGLE PRECISION ALPHA X PLUS Y (SAXPY)

GPU SAXPY in multiple languages and libraries

Part of Basic Linear Algebra Subroutines (BLAS) Library

$$z = \alpha x + y$$

x, y, z : vector

α : scalar

SAXPY: OPENACC COMPILER DIRECTIVES

Parallel C Code

```
void saxpy(int n,
           float a,
           float *x,
           float *y)
{
#pragma acc kernels
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

Parallel Fortran Code

```
subroutine saxpy(n, a, x, y)
  real :: x(:), y(:), a
  integer :: n, i
!$acc kernels
  do i=1,n
    y(i) = a*x(i)+y(i)
  enddo
!$acc end kernels
end subroutine saxpy


...
! Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...
```

www.openacc.org

SAXPY: CUBLAS LIBRARY

Serial BLAS Code

```
int N = 1<<20;

...

// Use your choice of blas library

// Perform SAXPY on 1M elements
blas_saxpy(N, 2.0, x, 1, y, 1);
```

Parallel cuBLAS Code

```
int N = 1<<20;


cublasInit();
cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);

// Perform SAXPY on 1M elements
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);

cublasShutdown();
```

You can also call cuBLAS from Fortran, C++, Python, and other languages:
http://developer.nvidia.com/cublas

SAXPY: CUDA C

Standard C

```
void saxpy(int n, float a,
      float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

int N = 1<<20;



// Perform SAXPY on 1M elements
saxpy(N, 2.0, x, y);
```

Parallel C

```
__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] + y[i];
}

int N = 1<<20;
cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, d_x, d_y);

cudaMemcpy(y, d_y, N, cudaMemcpyDeviceToHost);
```

http://developer.nvidia.com/cuda-toolkit

SAXPY: THRUST C++ TEMPLATE LIBRARY

Serial C++ Code (with STL and Boost)

```
int N = 1<<20;
std::vector<float> x(N), y(N);

...



// Perform SAXPY on 1M elements
std::transform(x.begin(), x.end(),
               y.begin(), y.end(),
          2.0f * _1 + _2);
```

www.boost.org/libs/lambda

Parallel C++ Code

```
int N = 1<<20;
thrust::host_vector<float> x(N), y(N);

...

thrust::device_vector<float> d_x = x;
thrust::device_vector<float> d_y = y;


// Perform SAXPY on 1M elements
thrust::transform(d_x.begin(), d_x.end(),
                  d_y.begin(), d_y.begin(),
                  2.0f * _1 + _2);
```

http://thrust.github.com

SAXPY: CUDA FORTRAN

Standard Fortran

```
module mymodule contains
  subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    do i=1,n
      y(i) = a*x(i)+y(i)
    enddo
  end subroutine saxpy
end module mymodule

program main
  use mymodule
  real :: x(2**20), y(2**20)
  x = 1.0, y = 2.0

  ! Perform SAXPY on 1M elements
  call saxpy(2**20, 2.0, x, y)

end program main
```

Parallel Fortran

```
module mymodule contains
  attributes(global) subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    attributes(value) :: a, n
    i = threadIdx%x+(blockIdx%x-1)*blockDim%x
    if (i<=n) y(i) = a*x(i)+y(i)
  end subroutine saxpy
end module mymodule

program main
  use cudafor; use mymodule
  real, device :: x_d(2**20), y_d(2**20)
  x_d = 1.0, y_d = 2.0

  ! Perform SAXPY on 1M elements
  call saxpy<<<4096,256>>>(2**20, 2.0, x_d, y_d)

end program main
```

http://developer.nvidia.com/cuda-fortran

SAXPY: PYTHON

Standard Python

```
import numpy as np


def saxpy(a, x, y):
  return [a * xi + yi
          for xi, yi in zip(x, y)]

x = np.arange(2**20, dtype=np.float32)
y = np.arange(2**20, dtype=np.float32)




cpu_result = saxpy(2.0, x, y)
```

http://numpy.scipy.org

Numba: Parallel Python

```
import numpy as np
from numba import vectorize

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

N = 1048576

# Initialize arrays
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 onGPU
C = saxpy(2.0, X, Y)
```

https://numba.pydata.org

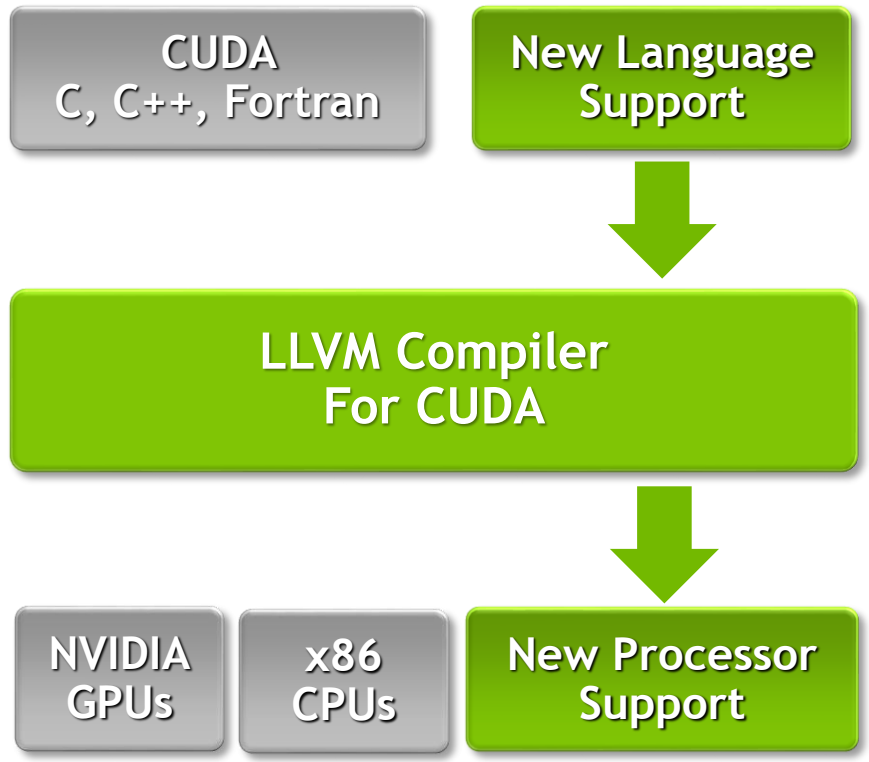
ENABLING ENDLESS WAYS TO SAXPY

Developers want to build front-ends for:

- Java, Python, R, DSLs

Target other processors like:

- ARM, FPGA, GPUs, x86



CUDA Compiler Contributed to Open Source LLVM

LLVM COMPILER INFRASTRUCTURE


GPU COMPUTING PLATFORM
FOR DL/ML

NVIDIA DEEP LEARNING SOFTWARE STACK

TRAINING

Training Data

Data Management

Training

Model Assessment

Trained Neural Network

Caffe2 | Chainer | Microsoft Cognitive Toolkit | mxnet | TensorFlow | PYTORCH | theano

INFERENCE

Data center

TensorRT

Embedded

JETPACK SDK

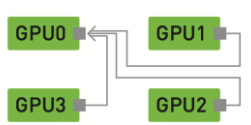
Automotive

DriveWorks SDK

NVIDIA DEEP LEARNING SDK and CUDA

cuDNN

NCCL

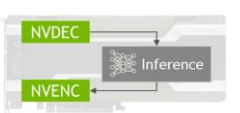


cuBLAS

TensorRT

cuSPARSE

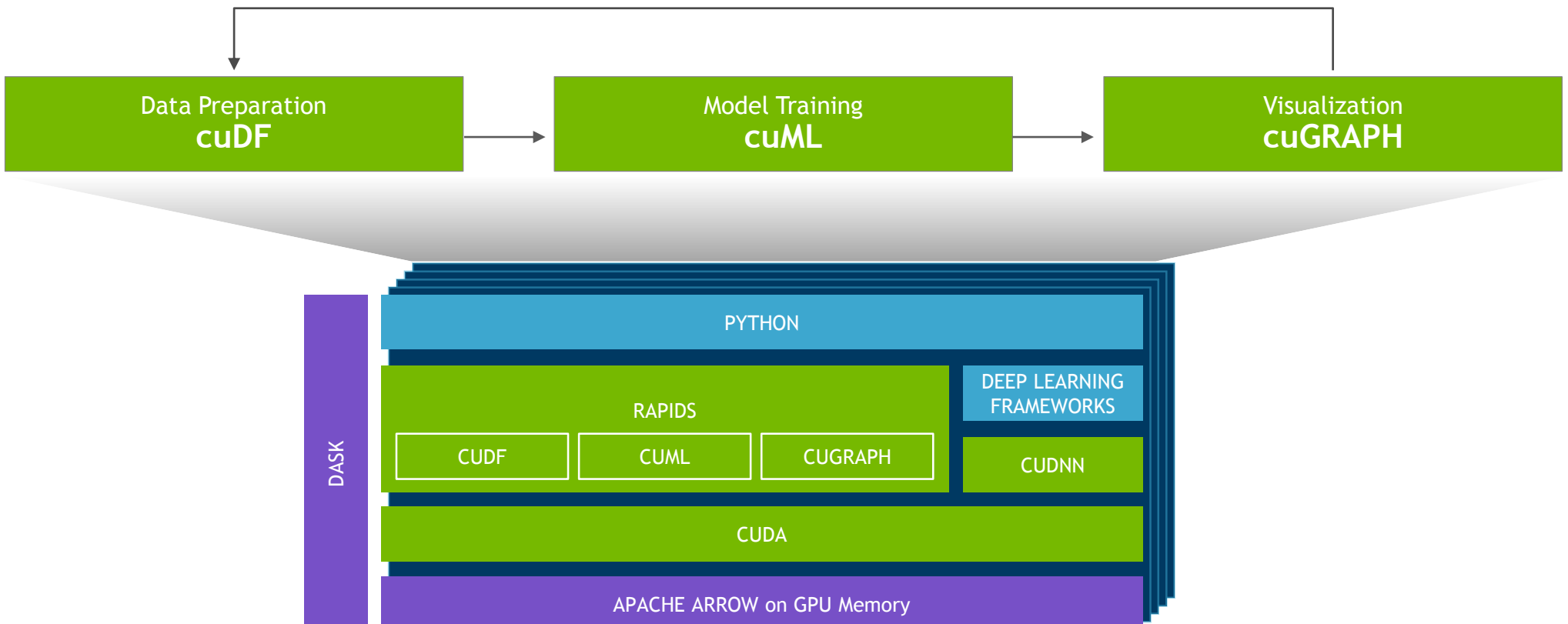
DeepStream SDK



developer.nvidia.com/deep-learning-software

RAPIDS — OPEN GPU DATA SCIENCE

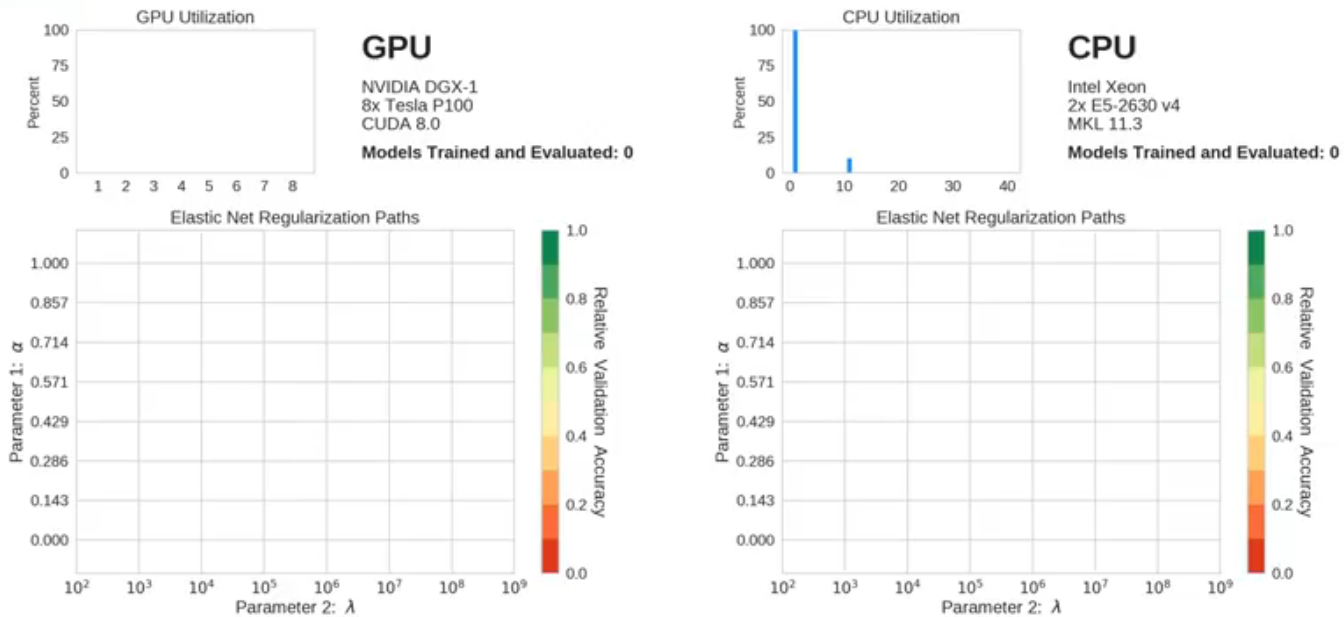
Software Stack Python



WHY RAPIDS

World's Fastest Machine Learning

H2O.ai Machine Learning – Generalized Linear Modeling

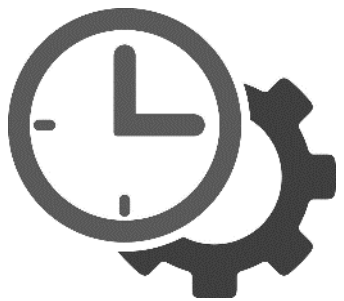


U.S. Census dataset (predict Income): 45k rows, 10k cols

Parameters: 5-fold cross-validation, $\alpha = \{\frac{i}{7}, i = 0 \ldots 7\}$, full λ-search


NGC FOR EFFICIENCY

CHALLENGES UTILIZING AI & HPC SOFTWARE

Installation	Optimization	Maintenance	Productivity
			
Complex, time consuming, and error-prone	Requires expertise to optimize framework performance	IT can’t keep up with frequent software upgrades	Users limited to older features and lower performance

NGC

The GPU-Optimized Software Hub

Simplify Deployments with Performance-optimized Containers

Innovate Faster with Ready-to-Use Solutions

Deploy Anywhere

https://www.nvidia.com/gpu-cloud/

SUMMARY

- Full Stack Optimization is key to performance
- Multiple choices for programming on GPU
- One is not an alternative to other. They co-exisit
- Universal hardware with Software stack is key to GPU computing


NVIDIA
Thank You


BACKUP

APPLICATION ACCELERATION STACKS

Breadth of Accelerated Apps Maximizes Data Center

Throughput, Utilization, Efficiency

HPC

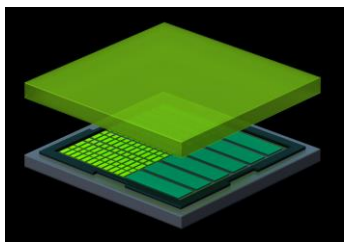
DATA ANALYTICS

DEEP LEARNING

MACHINE LEARNING

HYPERSCALE INFERENCE

RENDERING & VIZ



CUDA

GPU

https://www.nvidia.com/gpu-cloud/

PGI — THE NVIDIA HPC SDK

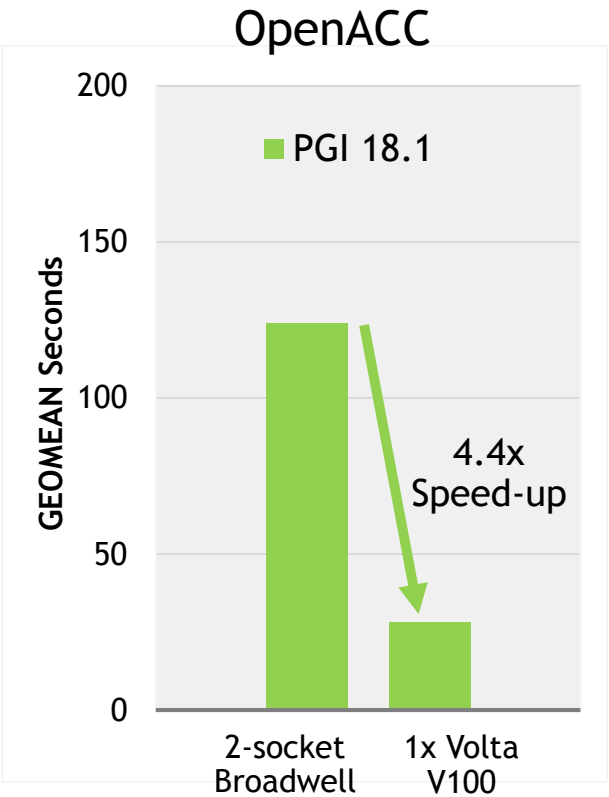
- Fortran, C & C++ Compilers
  - Optimizing, SIMD Vectorizing, OpenMP
- Accelerated Computing Features
  - CUDA Fortran, OpenACC Directives
- Multi-Platform Solution
  - X86-64 and OpenPOWER Multicore CPUs
  - NVIDIA Tesla GPUs
  - Supported on Linux, macOS, Windows
- MPI/OpenMP/OpenACC Tools
  - Debugger
  - Performance Profiler
  - Interoperable with DDT, TotalView

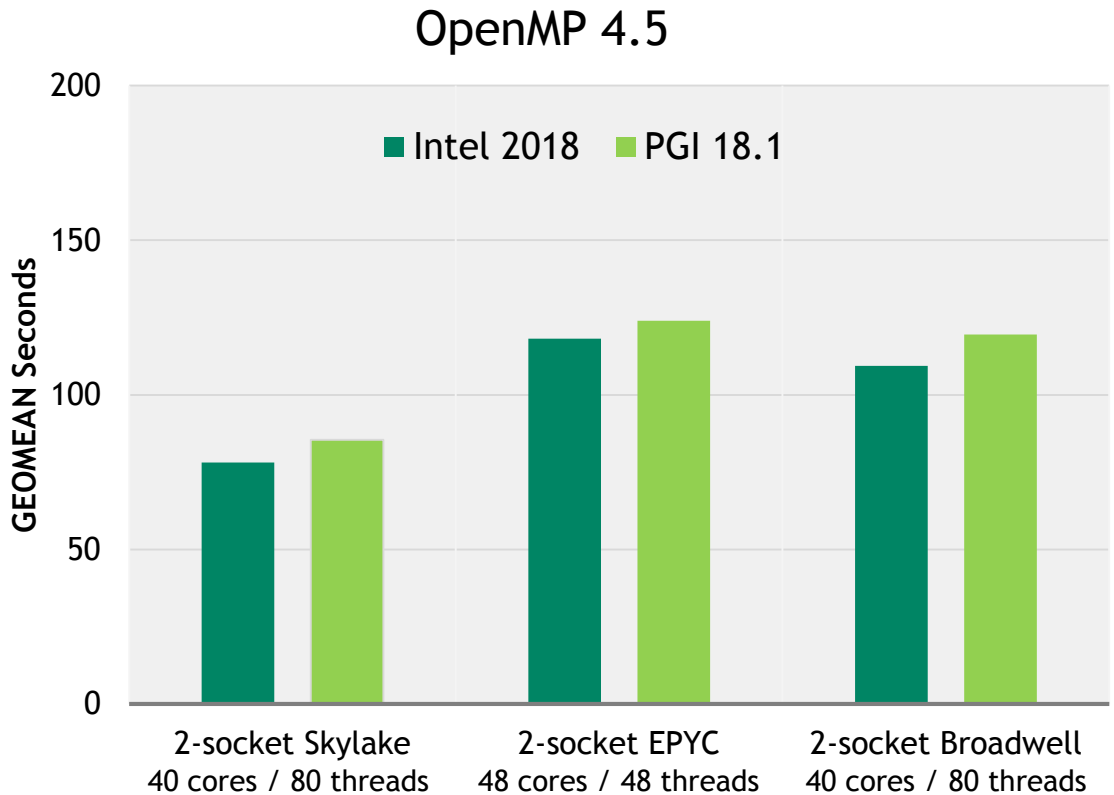
NVIDIA

PGI®

The Compilers & Tools for Supercomputing

SPEC ACCEL 1.2 BENCHMARKS





Performance measured February, 2018. Skylake: Two 20 core Intel Xeon Gold 6148 CPUs @ 2.4GHz w/ 376GB memory, hyperthreading enabled. EPYC: Two 24 core AMD EPYC 7451 CPUs @ 2.3GHz w/ 256GB memory. Broadwell: Two 20 core Intel Xeon E5-2698 v4 CPUs @ 3.6GHz w/ 256GB memory, hyperthreading enabled. Volta: NVIDIA DGX1 system with two 20 core Intel Xeon E5-2698 v4 CPUs @ 2.20GHz, 256GB memory, one NVIDIA Tesla V100-SXM2 GPU @ 1.53GHz. SPEC® is a registered trademark of the Standard Performance Evaluation Corporation (www.spec.org).

SINGLE CODE FOR MULTIPLE PLATFORMS

OpenACC - Performance Portable Programming Model for HPC

OpenPOWER

Sunway

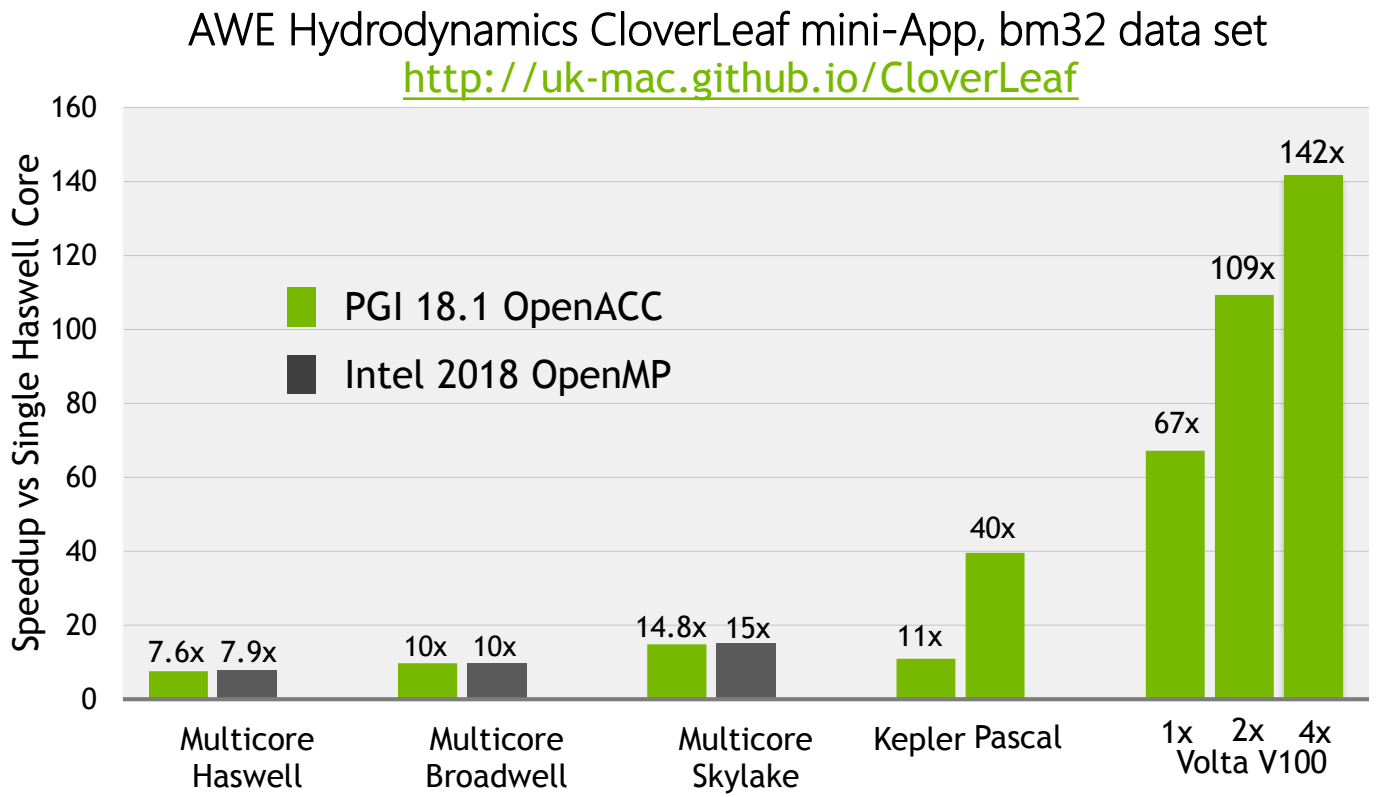
x86 CPU

x86 Xeon Phi

NVIDIA GPU

AMD GPU

PEZY-SC



Systems: Haswell: 2x16 core Haswell server, four K80s, CentOS 7.2 (perf-hsw10), Broadwell: 2x20 core Broadwell server, eight P100s (dgx1-prd-01), Broadwell server, eight V100s (dgx07), Skylake 2x20 core Xeon Gold server (sky-4).
Compilers: Intel 2018.0.128, PGI 18.1
Benchmark: CloverLeaf v1.3 downloaded from http://uk-mac.github.io/CloverLeaf the week of November 7 2016; CloverlLeaf_Serial; CloverLeaf_ref (MPI+OpenMP); CloverLeaf_OpenACC (MPI+OpenACC)
Data compiled by PGI February 2018.

OPENACC.ORG RESOURCES

Guides ● Talks ● Tutorials ● Videos ● Books ● Spec ● Code Samples ● Teaching Materials ● Events ● Success Stories ● Courses ● Slack ● Stack Overflow

OpenACC
Now in GCC

slack

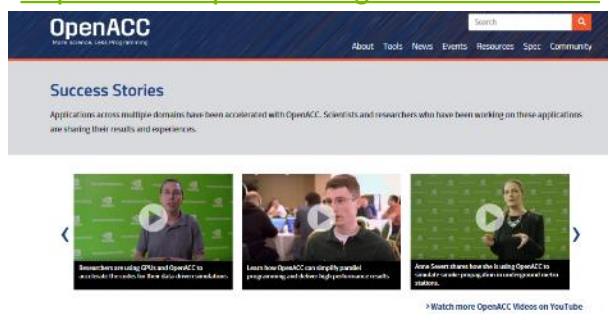
https://www.openacc.org/community#slack

Resources

https://www.openacc.org/resources

Success Stories

https://www.openacc.org/success-stories

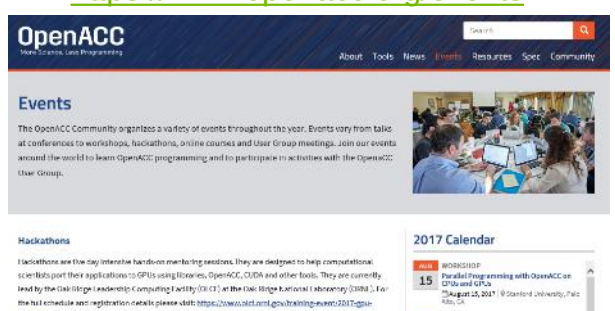


Compilers and Tools

https://www.openacc.org/tools

Events

https://www.openacc.org/events



NVIDIA

PGI COMPILERS FOR EVERYONE

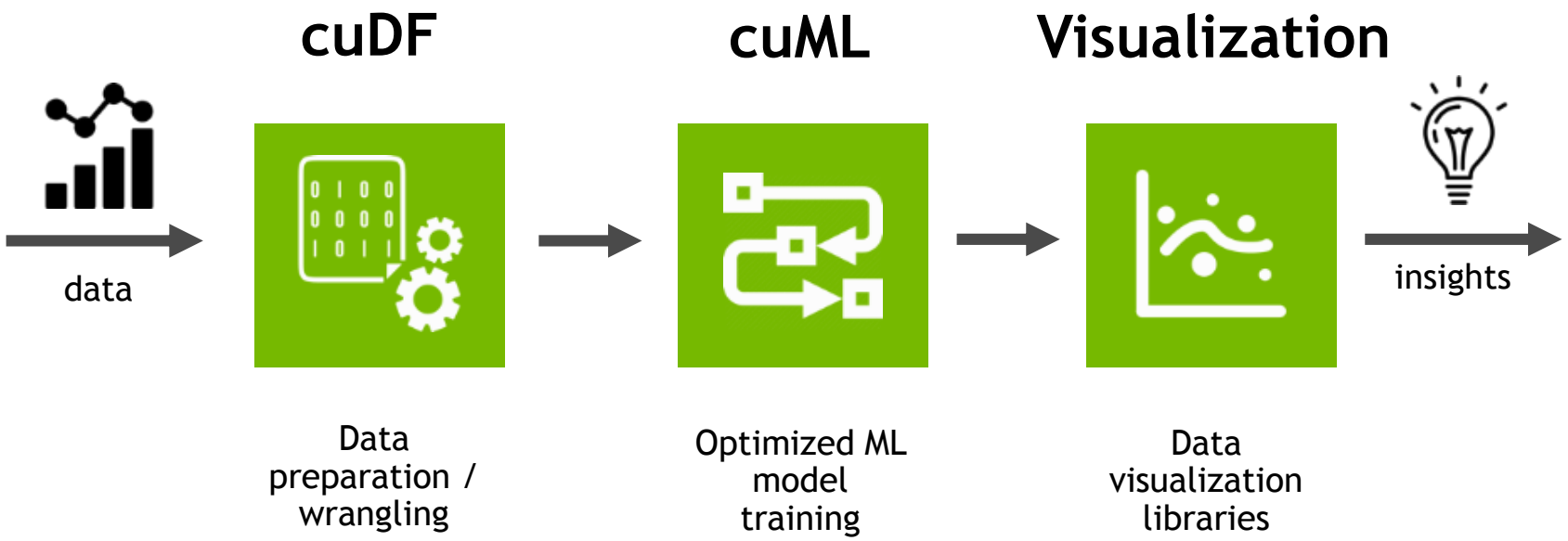
The PGI 18.10 Community Edition

FREE

	PGI Community Edition	PGI Professional Edition	PGI Enterprise Edition
PROGRAMMING MODELS OpenACC, CUDA Fortran, OpenMP, C/C++/Fortran Compilers and Tools	✓	✓	✓
PLATFORMS X86, OpenPOWER, NVIDIA GPU	✓	✓	✓
UPDATES	1-2 times a year	6-9 times a year	6-9 times a year
SUPPORT	User Forums	PGI Support	PGI Premier Services
LICENSE	Annual	Perpetual	Volume/Site

RE-IMAGINING DATA SCIENCE WORKFLOW

Open Source, End-to-end GPU-accelerated Workflow Built On CUDA



ACCELERATING MACHINE LEARNING

The RAPIDS Ecosystem

Open Source Community

ANACONDA. · Apache Arrow · UNROCK · PyData · QUANTSIGHT · learn · URSA LABS

Enterprise Data Science Platforms

IBM · ORACLE · sas · SAP

Startups

BLAZINGDB · FASTDATA · graphistry · H_2O.ai · kinetica · MAPR · omnisci

Deep Learning Integration

Chainer · PYTORCH

RAPIDS

GPU Servers

cisco · DELL EMC · Hewlett Packard Enterprise · IBM · Lenovo

Storage Partners

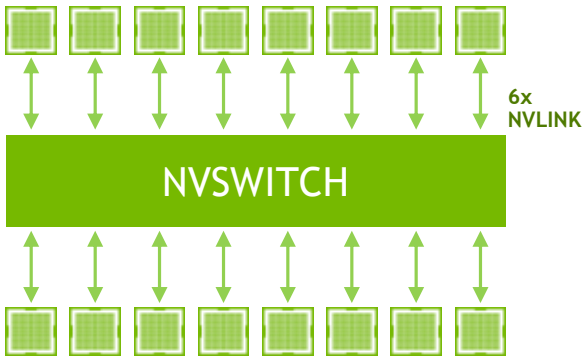
DELL EMC · IBM · NetApp · PURESTORAGE

PILLARS OF RAPIDS PERFORMANCE

CUDA Architecture

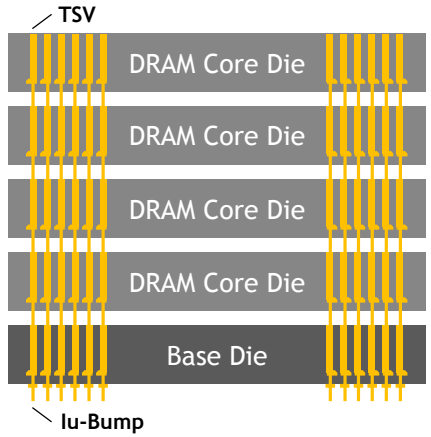
Massively parallel processing

NVLink/NVSwitch



High speed connecting between GPUs for distribute algorithms

Memory Architecture



Large virtual GPU memory, high-speed memory

NVIDIA cuDNN

Deep Learning Primitives

High performance building blocks for deep learning frameworks

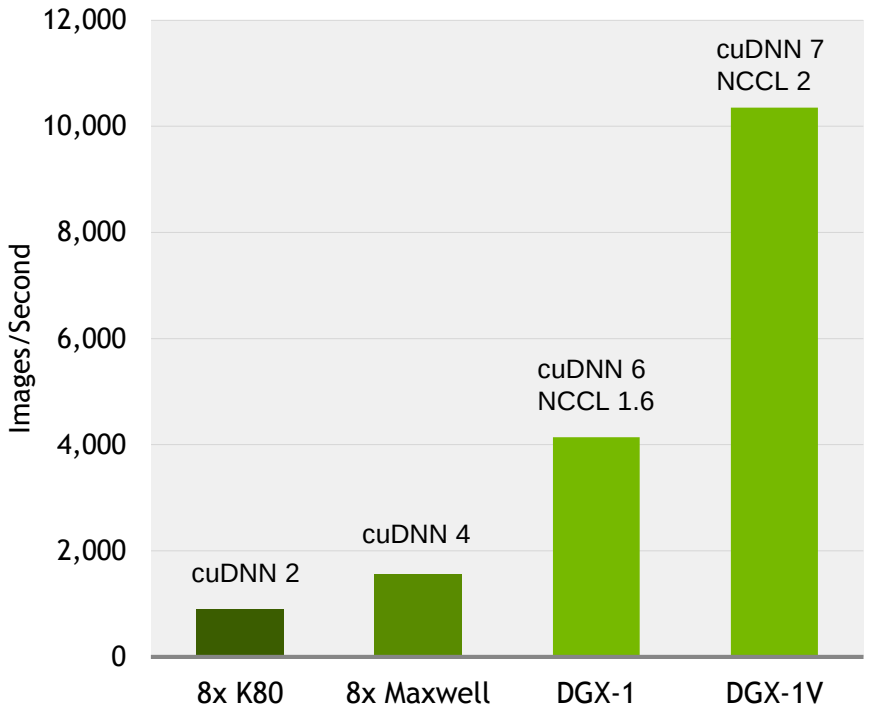
Drop-in acceleration for widely used deep learning frameworks such as Caffe2, Microsoft Cognitive Toolkit, PyTorch, Tensorflow and others

Accelerates industry vetted deep learning algorithms, such as convolutions, LSTM RNNs, fully connected, and pooling layers

Fast deep learning training performance tuned for NVIDIA GPUs

developer.nvidia.com/cudnn

Deep Learning Training Performance



"NVIDIA has improved the speed of cuDNN with each release while extending the interface to more operations and devices at the same time."

— Evan Shelhamer, Lead Caffe Developer, UC Berkeley

NVIDIA COLLECTIVE COMMUNICATIONS LIBRARY (NCCL)

Multi-GPU and Multi-node Collective Communication Primitives

Open-source High-performance multi-GPU and multi-node collective communication primitives optimized for NVIDIA GPUs

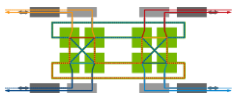
Fast routines for multi-GPU multi-node acceleration that maximizes inter-GPU bandwidth utilization

Easy to integrate and MPI compatible. Uses automatic topology detection to scale HPC and deep learning applications over PCIe and NVLink

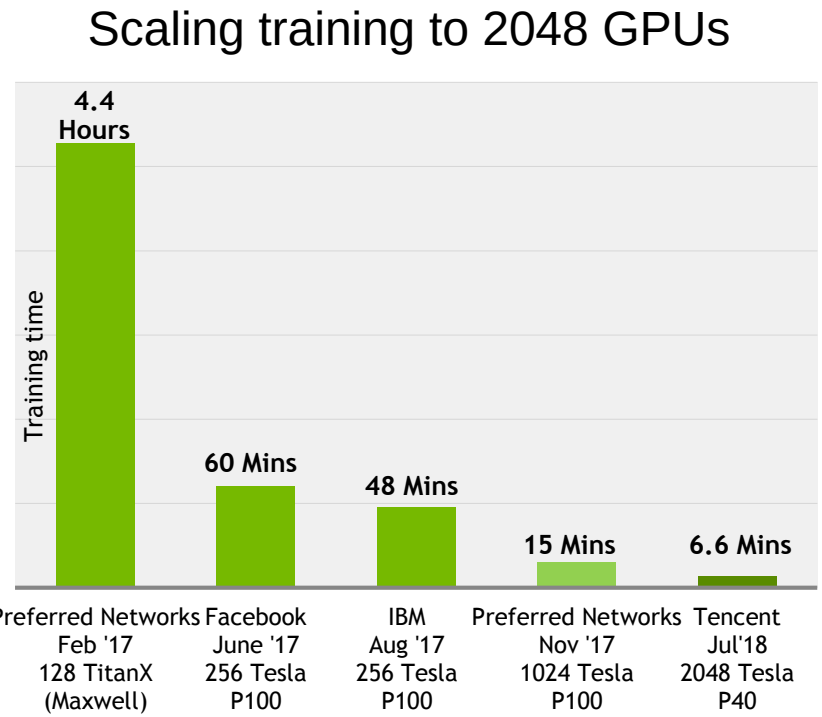
Accelerates leading deep learning frameworks such as Caffe2, Microsoft Cognitive Toolkit, MXNet, PyTorch and more

Multi-GPU:
NVLink, PCIe

Multi-Node:
InfiniBand verbs,
IP Sockets



Automatic
Topology
Detection



developer.nvidia.com/nccl

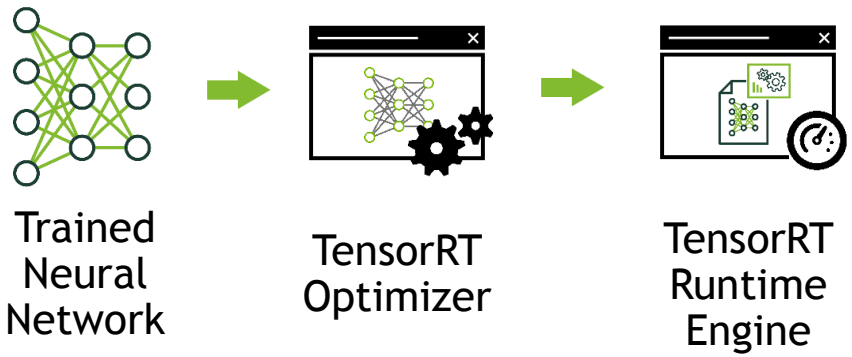
ResNet-50 | Dataset: Imagenet | Trained for 90 Epochs

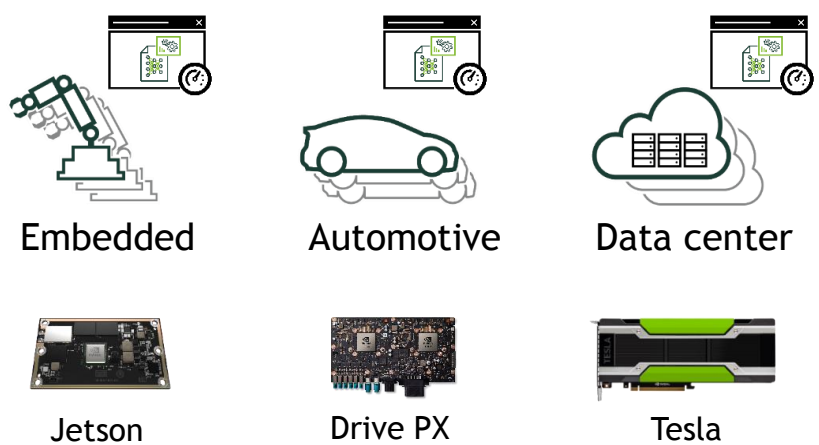
NVIDIA TensorRT

Deep Learning Inference Platform

Optimize and deploy neural networks in production environments

- Optimizer and runtime to maximize throughput for latency-critical services in production
- Deploy faster, more responsive and memory efficient applications with INT8 and FP16 optimizations
- Accelerates models trained in any framework with ONNX support and native framework integrations
- New TensorRT inference server





NVIDIA TensorRT INFERENCE SERVER

Containerized Microservice for Data Center Inference

Multiple models scalable across GPUs

Supports all popular AI frameworks

Seamless integration into DevOps deployments leveraging Docker and Kubernetes

Ready-to-run container, free from the NGC container registry