Introduction to the CUDA Platform

by NVIDIA Corporation

Source of presentation slides:
https://developer.nvidia.com/cuda-education

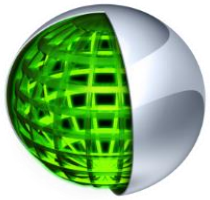
CUDA Parallel Computing Platform

www.nvidia.com/getcuda

Programming Approaches

- **Libraries** — "Drop-in" Acceleration
- **OpenACC Directives** — Easily Accelerate Apps
- **Programming Languages** — Maximum Flexibility

Development Environment



Nsight IDE
Linux, Mac and Windows
GPU Debugging and Profiling

CUDA-GDB debugger
NVIDIA Visual Profiler

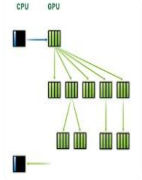
Open Compiler Tool Chain

Enables compiling new languages to CUDA platform, and CUDA languages to other architectures

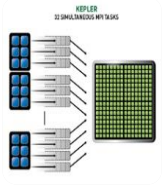
Hardware Capabilities

SMX

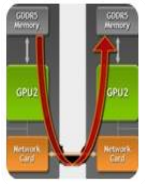
Dynamic Parallelism



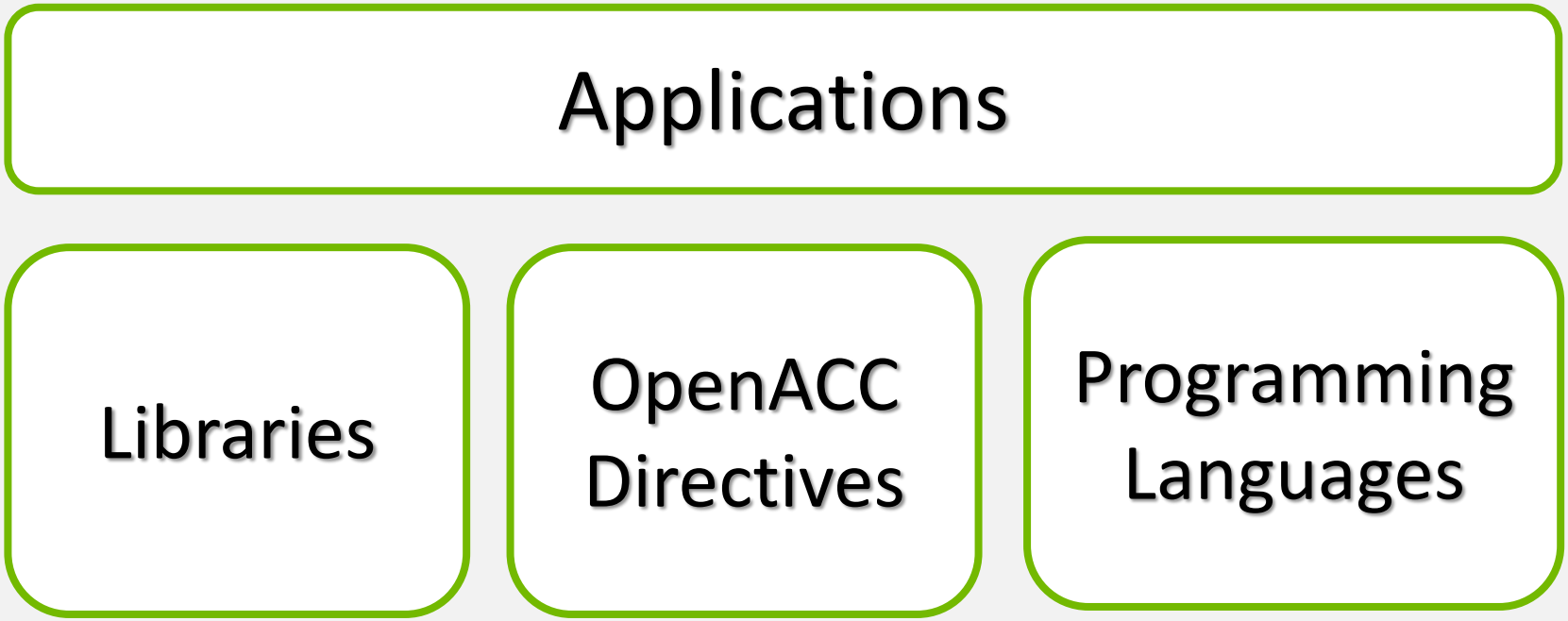
HyperQ



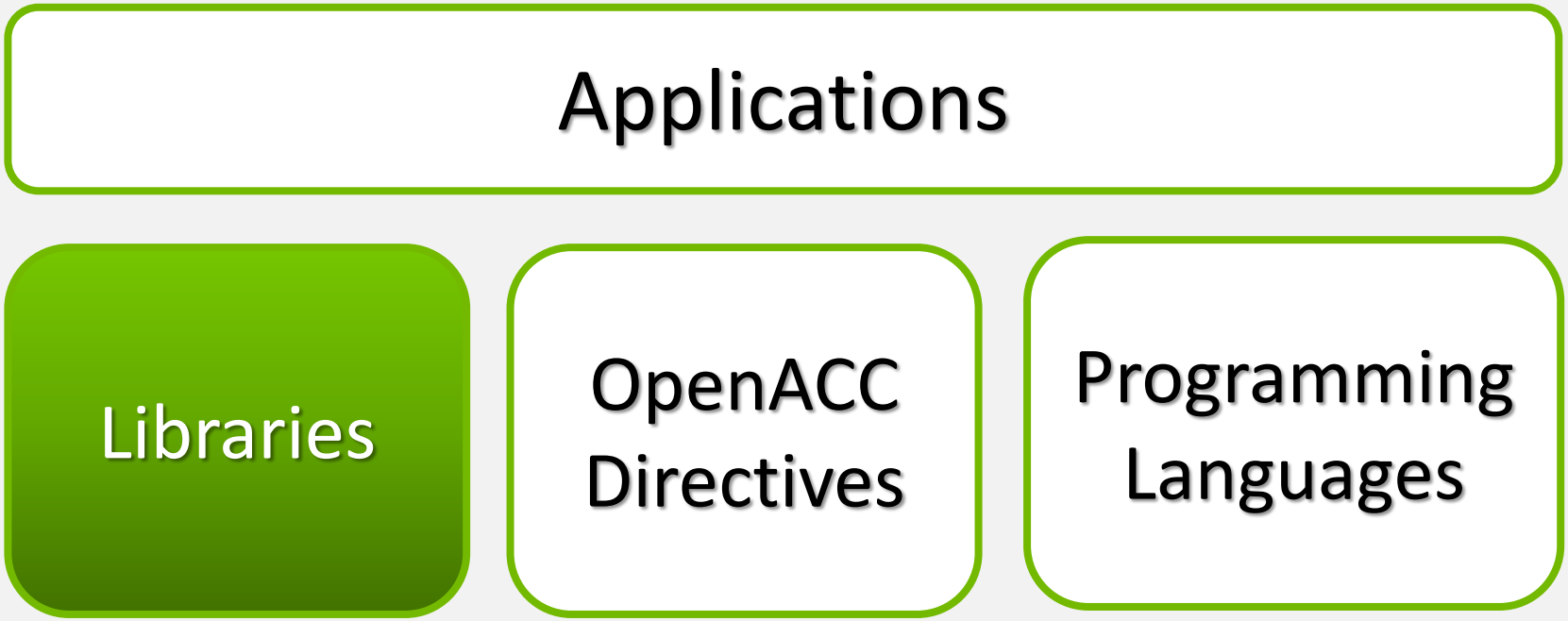
GPUDirect



3 Ways to Accelerate Applications



3 Ways to Accelerate Applications

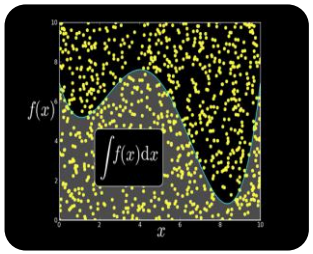


Libraries: Easy, High-Quality Acceleration

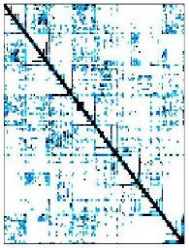
- **Ease of use:** Using libraries enables GPU acceleration without in-depth knowledge of GPU programming
- **"Drop-in":** Many GPU-accelerated libraries follow standard APIs, thus enabling acceleration with minimal code changes
- **Quality:** Libraries offer high-quality implementations of functions encountered in a broad range of applications
- **Performance:** NVIDIA libraries are tuned by experts

Some GPU-accelerated Libraries

NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE

NVIDIA NPP

GPU VSIPL

Vector Signal Image Processing

CULA tools

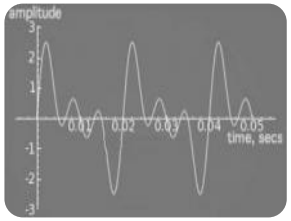
GPU Accelerated Linear Algebra

MAGMA

ICL UT

Matrix Algebra on GPU and Multicore

open source initiative



NVIDIA cuFFT

ROGUE WAVE SOFTWARE

IMSL Library

ArrayFire Matrix Computations

CUSP

Sparse Linear Algebra

open source initiative

C++ STL Features for CUDA

open source initiative

3 Steps to CUDA-accelerated application

- **Step 1:** Substitute library calls with equivalent CUDA library calls

  `saxpy ( … )` ► `cublasSaxpy ( … )`

- **Step 2:** Manage data locality

  - with CUDA: `cudaMalloc(), cudaMemcpy(), etc.`
  - with CUBLAS: `cublasAlloc(), cublasSetVector(), etc.`

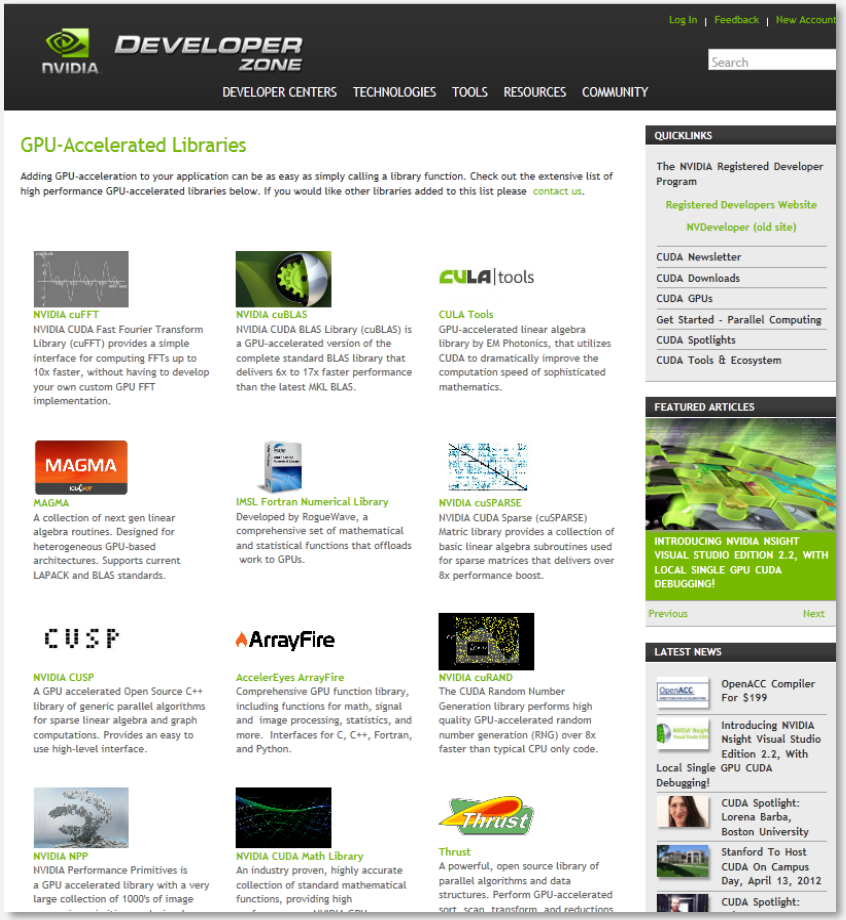
- **Step 3:** Rebuild and link the CUDA-accelerated library

```
nvcc myobj.o –l cublas
```

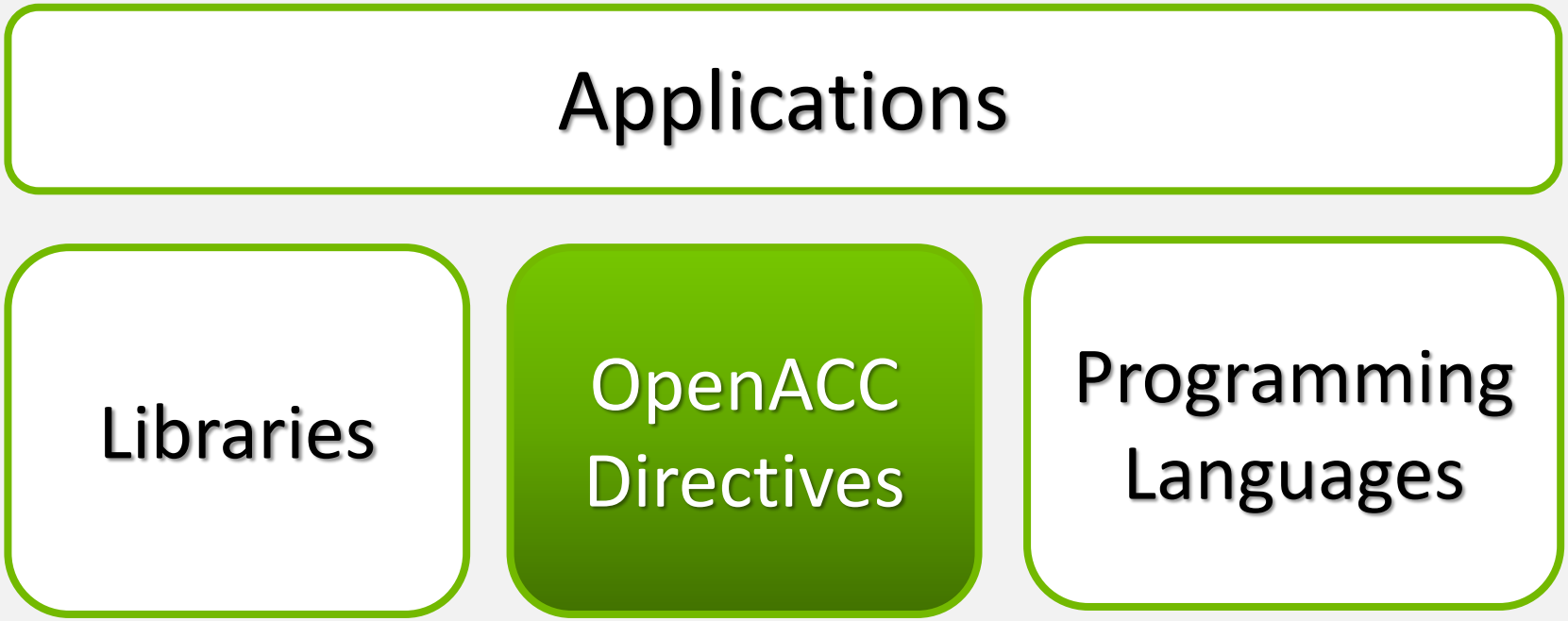
Explore the CUDA (Libraries) Ecosystem

- CUDA Tools and Ecosystem described in detail on NVIDIA Developer Zone:

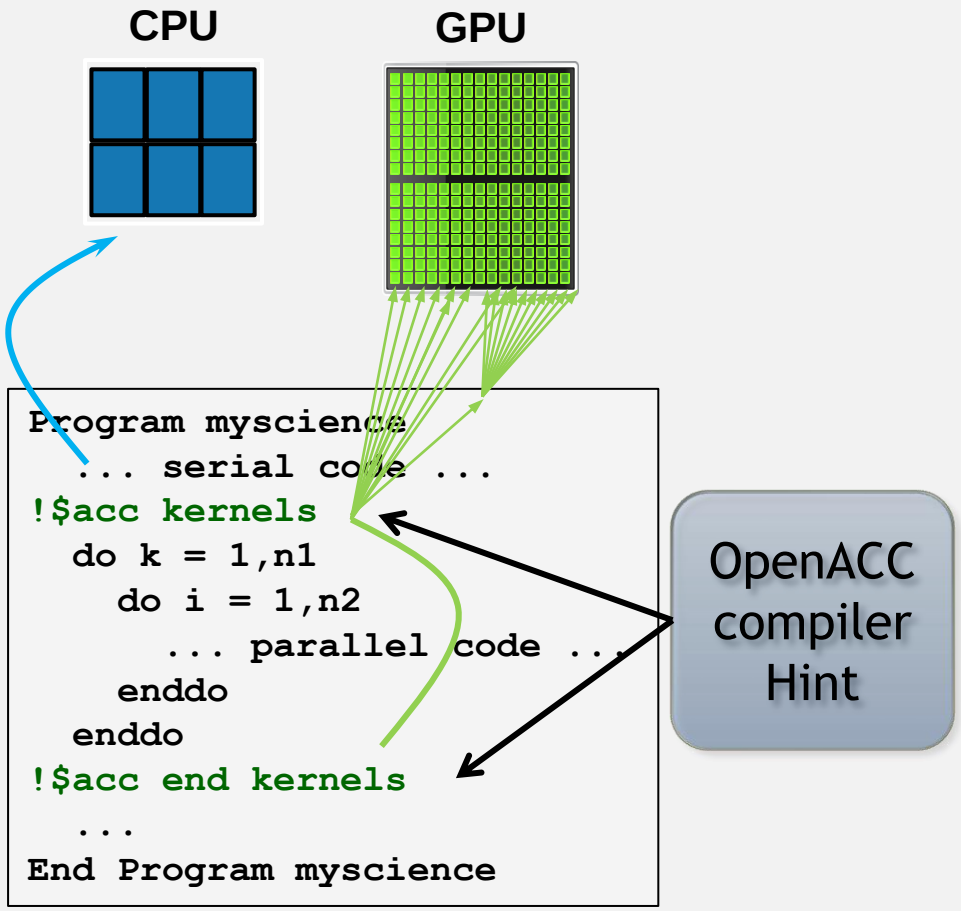
  developer.nvidia.com/cuda-tools-ecosystem



3 Ways to Accelerate Applications



OpenACC Directives



Simple Compiler hints

Compiler Parallelizes code

Works on many-core GPUs & multicore CPUs

OpenACC

OpenACC®
DIRECTIVES FOR ACCELERATORS

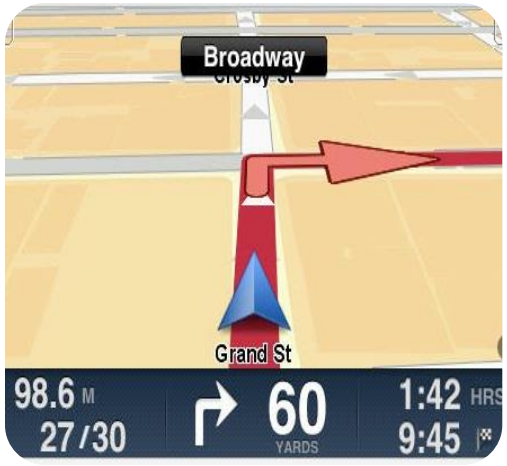
The Standard for GPU Directives

- **Easy:** Directives are the easy path to accelerate compute intensive applications
- **Open:** OpenACC is an open GPU directives standard, making GPU programming straightforward and portable across parallel and multi-core processors
- **Powerful:** GPU Directives allow complete access to the massive parallel power of a GPU

Directives: Easy & Powerful

Real-Time Object Detection

Global Manufacturer of Navigation Systems

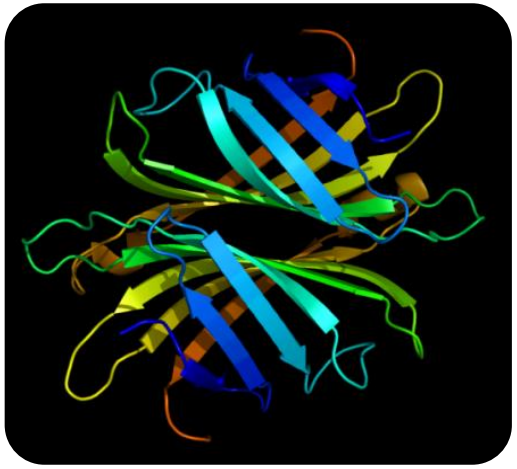


Valuation of Stock Portfolios using Monte Carlo

Global Technology Consulting Company

Interaction of Solvents and Biomolecules

University of Texas at San Antonio



5x in 40 Hours **2x in 4 Hours** **5x in 8 Hours**

"Optimizing code with directives is quite easy, especially compared to CPU threads or writing CUDA kernels. The most important thing is avoiding restructuring of existing code for production applications."

-- Developer at the Global Manufacturer of Navigation Systems

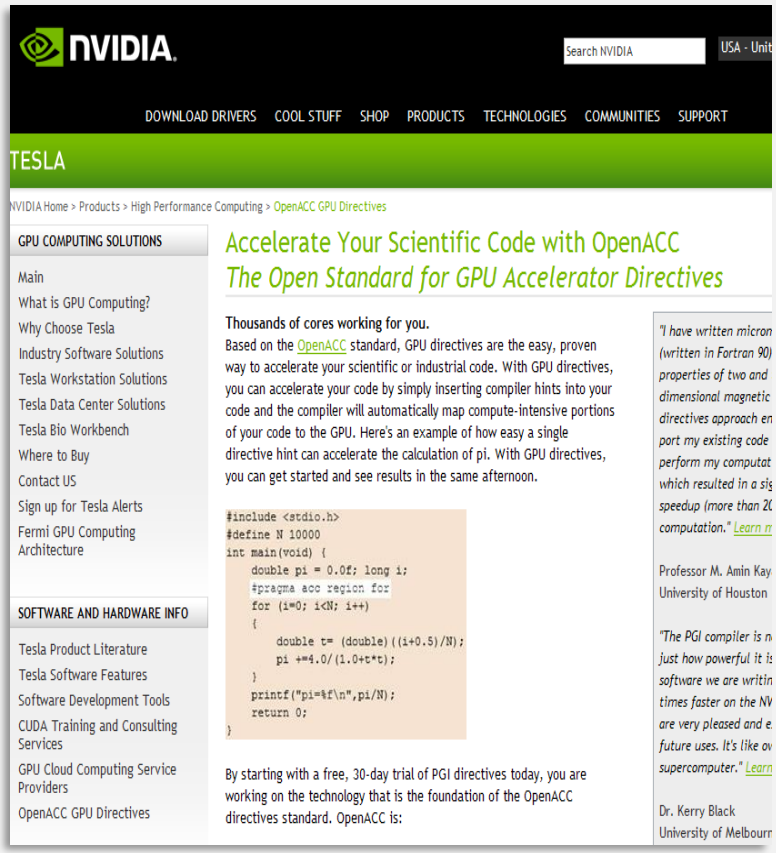
Start Now with OpenACC Directives

Free trial license to PGI Accelerator

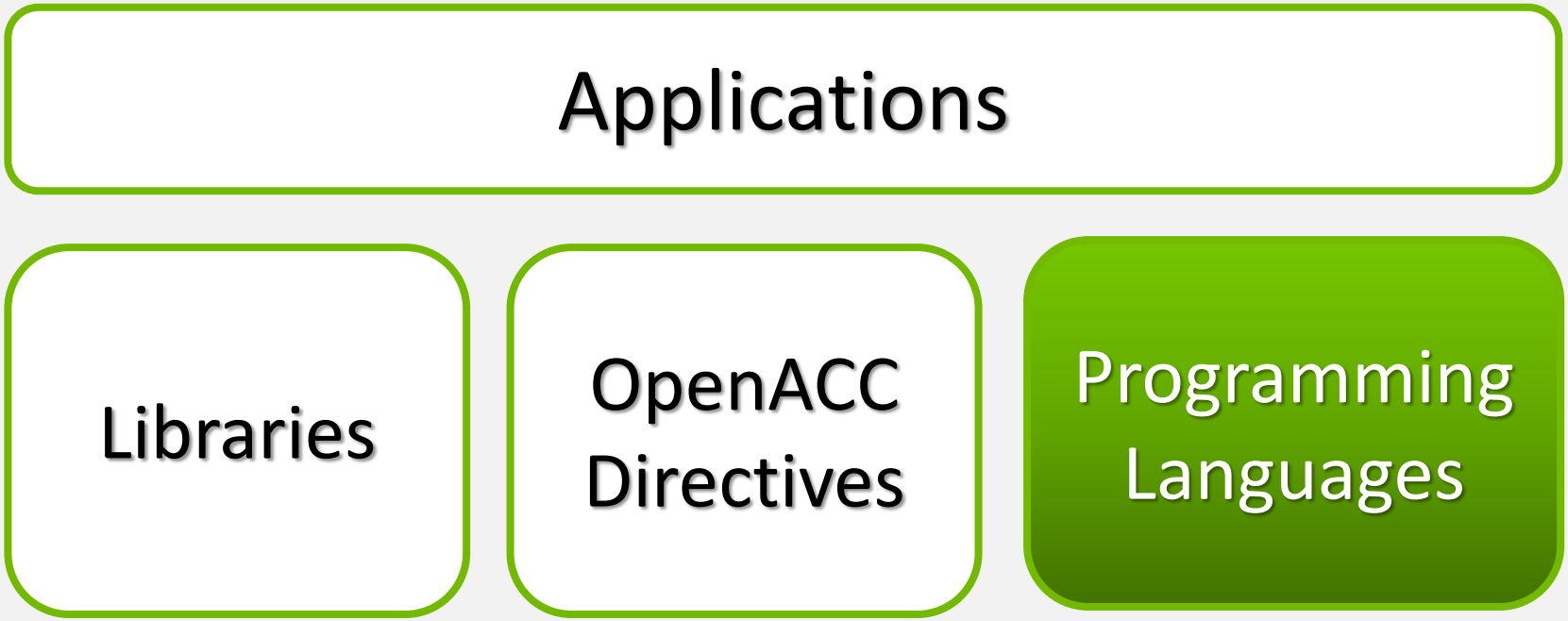
Tools for quick ramp

www.nvidia.com/gpudirectives

Sign up for a free trial of the directives compiler now!



3 Ways to Accelerate Applications



GPU Programming Languages

Numerical analytics	MATLAB, Mathematica, LabVIEW
Fortran	OpenACC, CUDA Fortran
C	OpenACC, CUDA C
C++	Thrust, CUDA C++
Python	PyCUDA, Copperhead
F#	Alea.cuBase

Rapid Parallel C++ Development

- Resembles C++ STL
- High-level interface
  - Enhances developer productivity
  - Enables performance portability between GPUs and multicore CPUs
- Flexible
  - CUDA, OpenMP, and TBB backends
  - Extensible and customizable
  - Integrates with existing software
- Open source

```
// generate 32M random numbers on host
thrust::host_vector<int> h_vec(32 << 20);
thrust::generate(h_vec.begin(),
                 h_vec.end(),
                 rand);

// transfer data to device (GPU)
thrust::device_vector<int> d_vec = h_vec;

// sort data on device
thrust::sort(d_vec.begin(), d_vec.end());

// transfer data back to host
thrust::copy(d_vec.begin(),
             d_vec.end(),
             h_vec.begin());
```

http://developer.nvidia.com/thrust or http://thrust.googlecode.com

Learn More

These languages are supported on all CUDA-capable GPUs.

You might already have a CUDA-capable GPU in your laptop or desktop PC!

CUDA C/C++
http://developer.nvidia.com/cuda-toolkit

Thrust C++ Template Library
http://developer.nvidia.com/thrust

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

PyCUDA (Python)
http://mathema.tician.de/software/pycuda

GPU.NET
http://tidepowerd.com

MATLAB
http://www.mathworks.com/discovery/matlab-gpu.html

Mathematica
http://www.wolfram.com/mathematica/new-in-8/cuda-and-opencl-support/

Getting Started

- Download CUDA Toolkit & SDK: www.nvidia.com/getcuda
- Nsight IDE (Eclipse or Visual Studio): www.nvidia.com/nsight
- Programming Guide/Best Practices:
  - docs.nvidia.com
- Questions:
  - NVIDIA Developer forums: devtalk.nvidia.com
  - Search or ask on: www.stackoverflow.com/tags/cuda
- General: www.nvidia.com/cudazone