



GPU Teaching Kit

Accelerated Computing



# Lecture 2.1 - Introduction to CUDA C

CUDA C vs. Thrust vs. CUDA Libraries

# Objective

- To learn the main venues and developer resources for GPU computing
  - Where CUDA C fits in the big picture

# 3 Ways to Accelerate Applications

Applications

Libraries

Easy to use  
Most Performance

Compiler  
Directives

Easy to use  
Portable code

Programming  
Languages

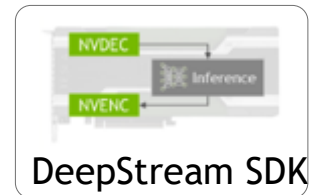
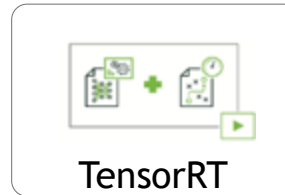
Most Performance  
Most Flexibility

# Libraries: Easy, High-Quality Acceleration

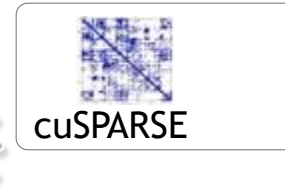
- **Ease of use:** Using libraries enables GPU acceleration without in-depth knowledge of GPU programming
- **“Drop-in”:** Many GPU-accelerated libraries follow standard APIs, thus enabling acceleration with minimal code changes
- **Quality:** Libraries offer high-quality implementations of functions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

# NVIDIA GPU Accelerated Libraries

## DEEP LEARNING



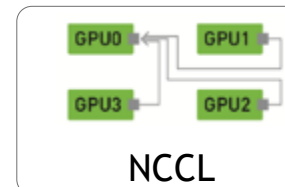
## LINEAR ALGEBRA



## SIGNAL, IMAGE, VIDEO



## PARALLEL ALGORITHMS



# Vector Addition in Thrust

```
#include <thrust/device_vector.h>
#include <thrust/copy.h>
```

```
int main(void) {
    size_t inputLength = 500;
    thrust::host_vector<float> hostInput1(inputLength);
    thrust::host_vector<float> hostInput2(inputLength);
    thrust::device_vector<float> deviceInput1(inputLength);
    thrust::device_vector<float> deviceInput2(inputLength);
    thrust::device_vector<float> deviceOutput(inputLength);
```

```
    thrust::copy(hostInput1.begin(), hostInput1.end(), deviceInput1.begin());
    thrust::copy(hostInput2.begin(), hostInput2.end(), deviceInput2.begin());
```

```
    thrust::transform(deviceInput1.begin(), deviceInput1.end(),
                      deviceInput2.begin(), deviceOutput.begin(),
                      thrust::plus<float>());
```

```
}
```

# Compiler Directives: Easy, Portable Acceleration

- **Ease of use:** Compiler takes care of details of parallelism management and data movement
- **Portable:** The code is generic, not specific to any type of hardware and can be deployed into multiple languages
- **Uncertain:** Performance of code can vary across compiler versions

# OpenACC

- Compiler directives for C, C++, and FORTRAN

```
#pragma acc parallel loop  
copyin(input1[0:inputLength],input2[0:inputLength]),  
copyout(output[0:inputLength])  
for(i = 0; i < inputLength; ++i) {  
    output[i] = input1[i] + input2[i];  
}
```



# Programming Languages: Most Performance and Flexible Acceleration

- **Performance:** Programmer has best control of parallelism and data movement
- **Flexible:** The computation does not need to fit into a limited set of library patterns or directive types
- **Verbose:** The programmer often needs to express more details

# GPU Programming Languages

**Numerical analytics** ►

MATLAB, Mathematica, LabVIEW

**Python** ►

PyCUDA, Numba

**Fortran** ►

CUDA Fortran, OpenACC

**C** ►

CUDA C, OpenACC

**C++** ►

CUDA C++, Thrust

**C#** ►

Hybridizer

# CUDA - C

Applications

Libraries

Compiler  
Directives

Programming  
Languages

Easy to use  
Most Performance

Easy to use  
Portable code

**Most Performance**  
**Most Flexibility**



# GPU Teaching Kit

Accelerated Computing



The GPU Teaching Kit is licensed by NVIDIA and the University of Illinois under the [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).