GPU Teaching Kit

Accelerated Computing

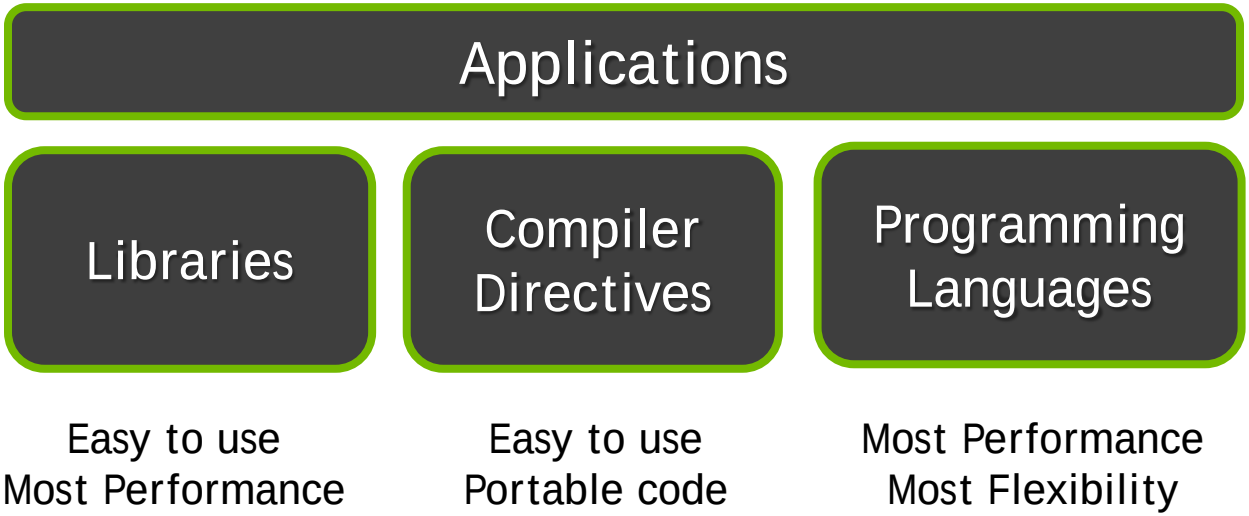
Lecture 2.1 - Introduction to CUDA C

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



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

GPU Accelerated Libraries

Linear Algebra
FFT, BLAS,
SPARSE, Matrix

Numerical & Math
RAND, Statistics

Data Struct. & AI
Sort, Scan, Zero Sum

Visual Processing
Image & Video

NVIDIA
Video
Encode

Vector Addition in Thrust

```
thrust::device_vector<float> deviceInput1(inputLength);
thrust::device_vector<float> deviceInput2(inputLength);
thrust::device_vector<float> deviceOutput(inputLength);

thrust::copy(hostInput1, hostInput1 + inputLength,
    deviceInput1.begin());
thrust::copy(hostInput2, hostInput2 + inputLength,
    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
Fortran	CUDA Fortran
C	CUDA C
C++	CUDA C++
Python	PyCUDA, Copperhead, Numba
F#	Alea.cuBase

CUDA - C

Easy to use
Most Performance

Easy to use
Portable code

Most Performance
Most Flexibility

GPU Teaching Kit

Accelerated Computing