# CUDA C++

5 JUNE 2024 | JAN H. MEINKE

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum

# CUDA AND C++

- CUDA host code has been compiled as C++ code since version 2!
- Some C++ features, e.g., support for templates since CUDA 1.x
- C++ 11 features supported in host *and* device code since CUDA 7
- C++ 14 features supported in host *and* device code since CUDA 9
- C++ 17 features supported in host *and* device code since CUDA 11
- C++ 20 features supported in host *and* device code since CUDA 12
- pSTL supported on GPU with NVHPC toolkit

# A SAMPLE OF C++ 11 FEATURES

***auto***

***template***

**memory management**

**range-based for loops**

**lambdas**

# WRITING KERNELS FOR DIFFERENT DATA TYPES

```
__global__ void saxpy(float alpha, float* x, float* y, size_t n){
    auto i = blockDim.x * blockIdx.x + threadIdx.x;
    if(i < n){
        y[i] = alpha * x[i] + y[i];
    }
}
```

# WRITING KERNELS FOR DIFFERENT DATA TYPES

```
__global__ void daxpy(double alpha, double* x, double* y, size_t n){
    auto i = blockDim.x * blockIdx.x + threadIdx.x;
    if(i < n){
        y[i] = alpha * x[i] + y[i];
    }
}
```

# WRITING KERNELS FOR DIFFERENT DATA TYPES

```
template <typename T>
__global__ void axpy(T alpha, T* x, T* y, size_t n){
    auto i = blockDim.x * blockIdx.x + threadIdx.x;
    if(i < n){
        y[i] = alpha * x[i] + y[i];
    }
}
```

# Exercise

05-CUDA_C++/exercises/tasks/gemm

Compile with make.

# STRUCT INSTEAD OF RAW POINTER

```
struct Matrix {
Matrix(int h, int w): height(h), width(w) {
    cudaMallocManaged(&data, height *
width *sizeof(double));
};
~Matrix(){
    cudaFree(data);
}
int height;
int width;
int* data;
};
```

You can pass structs to kernels
Data members are trivially copyable
Free is called automatically

```
__global__
void mm(Matrix A, Matrix B, Matrix C);

Matrix A(1024, 1024);
…
mm<<<...>>>(A, B, C);
```

# TRANSPARENT TYPES

```
class Managed {
public:
  void *operator new(size_t len) {
    void *ptr;
    cudaMallocManaged(&ptr, len);
    cudaDeviceSynchronize();
    return ptr;
  }

  void operator delete(void *ptr) {
    cudaDeviceSynchronize();
    cudaFree(ptr);
  }
};
```

Closely modeled after “Unified Memory in CUDA 6” (see Refs)

# TRANSPARENT TYPES

```
template <class T>
class Array : public Managed {
    size_t n;
    T* data;

public:
  Array (const Array &a) {
    n = a.n;
    cudaMallocManaged(&data, n);
    memcpy(data, a.data, n);
  }
   // Also have to implement operator[], for example
};
```

# TRANSPARENT TYPES

```
// Pass-by-reference version
__global__ void kernel_by_ref(Array &data) { ... }
// Pass-by-value version
__global__ void kernel_by_val(Array data) { ... }

int main(void) {
  Array *a = new Array;
  ...
  // pass data to kernel by reference
  kernel_by_ref<<<1,1>>>(*a);
  // pass data to kernel by value -- this will create a copy
  kernel_by_val<<<1,1>>>(*a);
}
```

# THRUST ON DEVICE

```
__global__
void xyzw_frequency_thrust_device(int *count, char *text, int n)
{
  const char letters[] { 'x','y','z','w' };

  *count = thrust::count_if(thrust::device, text, text+n, [=](char c) {
    for (const auto x : letters)
      if (c == x) return true;
    return false;
  });
}
```

# THE STANDARD TEMPLATE LIBRARY (STL)

**array**

**sort**

***vector***

**transform**

**...**

**for_each**

**reduce**

**list**

**accumulate**

# THE STANDARD TEMPLATE LIBRARY (STL)

Templates

- Allow different type

Iterators

- Generic algorithms

# LIBCU++

Implementation of *some* STL features, e.g.,

- atomic <cuda/std/atomic>
- complex <cuda/std/complex>
- chrono <cuda/std/chrono>
- array <cuda/std/array>
- span <cuda/std/span>
- mdspan (soon)
- …

Header-only library with host and device functions

Comes with CUDA SDK and NVHPC SDK

Included in standard include path → no compiler options needed

https://nvidia.github.io/libcudacxx/

# STD::SPAN

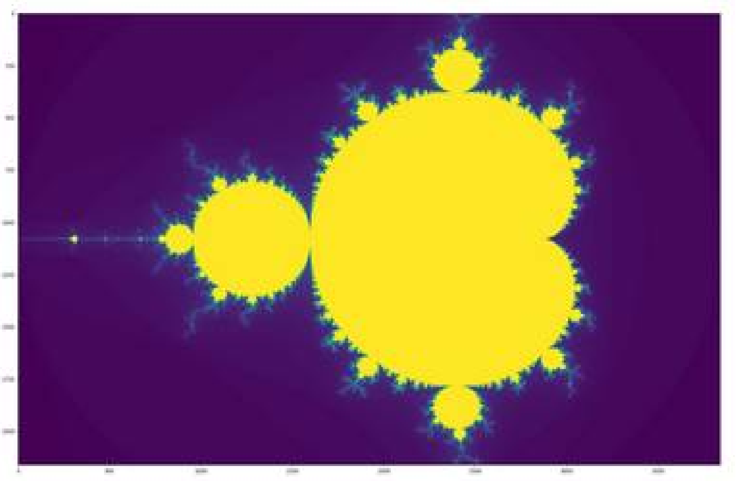
- View of contiguous memory
- Knows its own size
- Access through operator[]
- Device aware version in cuda::std::span

```
template <class T>
__global__ void foo(cuda::std::span<T> x){
  auto i = threadIdx.x + blockIdx.x * blockDim.x;
  if (i < x.size()){
    x[i] = …;
  }
}
auto main() → int {
  double* x = nullptr;
  std::vector<double, managedAlloc> y(10000, 2.7);
  cudaMallocManaged(&x, sizeof(double) * 12000);
  foo<<<40, 256>>>(y);
  foo<<<47, 256>>>({x, 12000});
...
```

# Exercise

05-CUDA_C++/exercises/tasks/axpy

Compile with make.



# Exercise

05-CUDA_C++/exercises/tasks/mandelbrot

Compile with nvcc mandelbrot.cu -o mandelbrot.
Launch with $JSC_SUBMIT_CMD ./mandelbrot.

# AN STL EXAMPLE

```
#include <algorithm>
#include <numeric>
#include <iostream>
#include <vector>

int main(){
    size_t N = 10'000;
    std::vector x(N, 1.0 / N);
    std::cout << "The sum of the elements of x is " << std::reduce(x.begin(), x.end(),
0.0);
}
```

# PARALLEL STL (PSTL)

**execution::par**

***sort***

***execution::unseq***

***transform***

**execution::seq**

***for_each***

***reduce***

**execution::par_unseq**

**accumulate**

https://en.cppreference.com/w/cpp/algorithm

# A PSTL EXAMPLE

```
#include <execution>
#include <iostream>
#include <numeric>
#include <vector>

int main(){
    size_t N = 10'000;
    std::vector x(N, 1.0 / N);
    std::cout << "The sum of the elements of x is " <<
       std::reduce(std::execution::par_unseq, x.begin(), x.end(), 0.0);
}
```

Much more of this
on
Friday

# REFERENCES

- C++11 in CUDA: Variadic Templates - https://developer.nvidia.com/blog/cplusplus-11-in-cuda-variadic-templates
- managed_allocator/README.md at master · jaredhoberock/managed_allocator · GitHub - https://github.com/jaredhoberock/managed_allocator/blob/master/README.md
- Unified Memory in CUDA 6 - https://developer.nvidia.com/blog/unified-memory-in-cuda-6

# REFERENCES

- Unified Memory in CUDA 6 - https://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6
- Faster Parallel Reductions on Kepler https://devblogs.nvidia.com/parallelforall/faster-parallel-reductions-kepler
- CUDA 7.5 https://devblogs.nvidia.com/parallelforall/new-features-cuda-7-5/
- CUDA 8.0 https://devblogs.nvidia.com/parallelforall/cuda-8-features-revealed/