

CUDA C++

28 APRIL 2022 | JAN H. MEINKE

CUDA AND C++

- CUDA host code has been compiled as C++ code since version 2!
- Some C++ features, e.g., templates have been supported 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
- 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] = a * 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] = a * 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

CUDA++/exercises/tasks/gemm

Compile with make.

THE STANDARD TEMPLATE LIBRARY (STL)

vector **array** **sort**

transform

... **for_each** **reduce**

list **accumulate**

THE STANDARD TEMPLATE LIBRARY (STL)

Templates

- Allow different type

Iterators

- Generic algorithms

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);
}
```

FUNCTION OBJECT (AKA FUNCTOR)

```
template <class T>
class In_range {
    const T val1;
    const T val2;
public:
    In_range(const T& v1, const T& v2) : val1(v1), val2(v2) {}
    bool operator()(const T& x) const {return (x >= val1 && x < val2);}
};
```

Can be used, e.g., in `std::count()`:

```
std::count_if(v.begin(), v.end(), In_range<int>(3, 6));
```

LAMDAS

```
auto lambda = [](const int& x){return (x >= 3 && x < 6);}
```

Can be used, e.g., in `std::count_if()`:

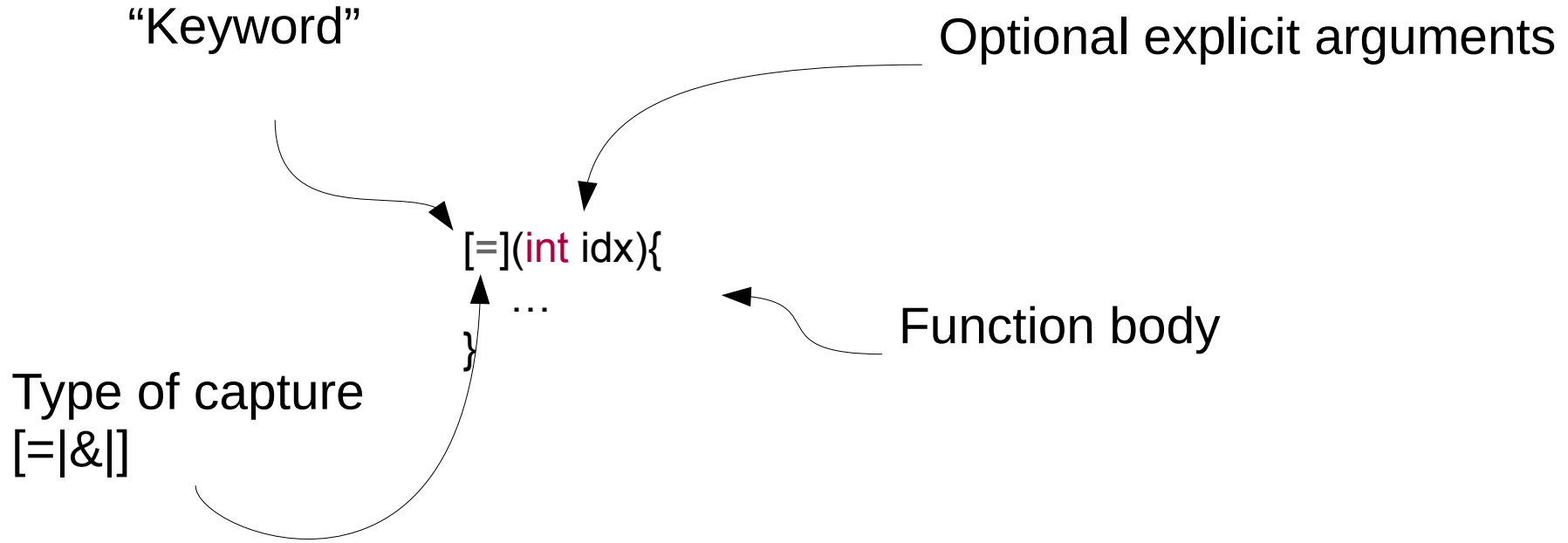
```
std::count_if(v.begin(), v.end(), [](const int& x){return (x >= 3 && x < 6);});
```

LAMDAS

```
std::vector<int> v {5, 1, 1, 3, 1, 4, 1, 3, 3, 2};  
int a = 3;  
int b = 6;  
auto lambda = [&](const int x){return (x >= a && x < b);};  
auto ct36 = std::count_if(v.begin(), v.end(), lambda);
```

LAMDAS

Lambdas are anonymous functions that can capture variables.



STD::TRANSFORM + LAMBDA

```
#include <algorithm>
#include <execution>
#include <vector>
```

```
Template <class T>
void scale_vector(std::vector<T> &&x, std::vector<T> &&y, T a) {
    std::transform(x.begin(), x.end(), y, [=](auto x) {
        return a * x;});
}
```

Exercise

CUDA++/exercises/tasks/transform

Compile with make.

COUNTING

Sometimes it's easier to use an index:

- Container of indices

```
std::vector idx(x.size(), 0);  
std::iota(idx.begin(), idx.end(), 0);  
std::for_each(idx.begin(), idx.end(), ...)
```

- Counting iterator (for example from thrust)

```
auto r = thrust::counting_iterator<int>(0);  
std::for_each(r, r + N, ...)
```

CATCHING POINTERS BY VALUE

Access to CPU memory not allocated with new → memory access error

Reference capture of scalars → use value capture instead

Value capture of vector can also lead to problems → use pointer instead

```
auto ptr_x = x.data();
```

Exercise

`CUDA++/exercises/tasks/for_each`

Compile with make.

TRANSFORM_REDUCE

Transformation (map) and reduction (reduce) are often combined.
C++ offers transform_reduce to do it in one call:

```
std::transform_reduce(x.begin(), x.end(), y.begin(),  
                    -1.0, [](auto a, auto b){return std::max(a, b);},  
                    [](auto a, auto b){ return std::abs(a - b);}  
                    );
```

First comes the **reduction** operation, **then** comes the **transform** operation.

Exercise

`CUDA++/exercises/tasks/jacobi`

Compile with make.

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;  
    });  
}
```

REFERENCES

- C++11 in CUDA: Variadic Templates -
<https://devblogs.nvidia.com/parallelforall/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
- C++11 Archives | Parallel Forall -
<https://devblogs.nvidia.com/parallelforall/tag/c11>
- The Saint on Porting C++ Classes to CUDA with Unified Memory -
<https://devblogs.nvidia.com/parallelforall/the-saint-porting-c-classes-cuda-unified-memory>

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/>

REFERENCES

- Accelerating Standard C++ with GPUs Using stdpar,
<https://developer.nvidia.com/blog/accelerating-standard-c-with-gpus-using-stdpar/>