GPU PROGRAMMING WITH STANDARD C++

7 JUNE 2024 | JAN H. MEINKE

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

AN STL EXAMPLE

```cpp
#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

```cpp
#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));
```

LAMBDAS

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

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

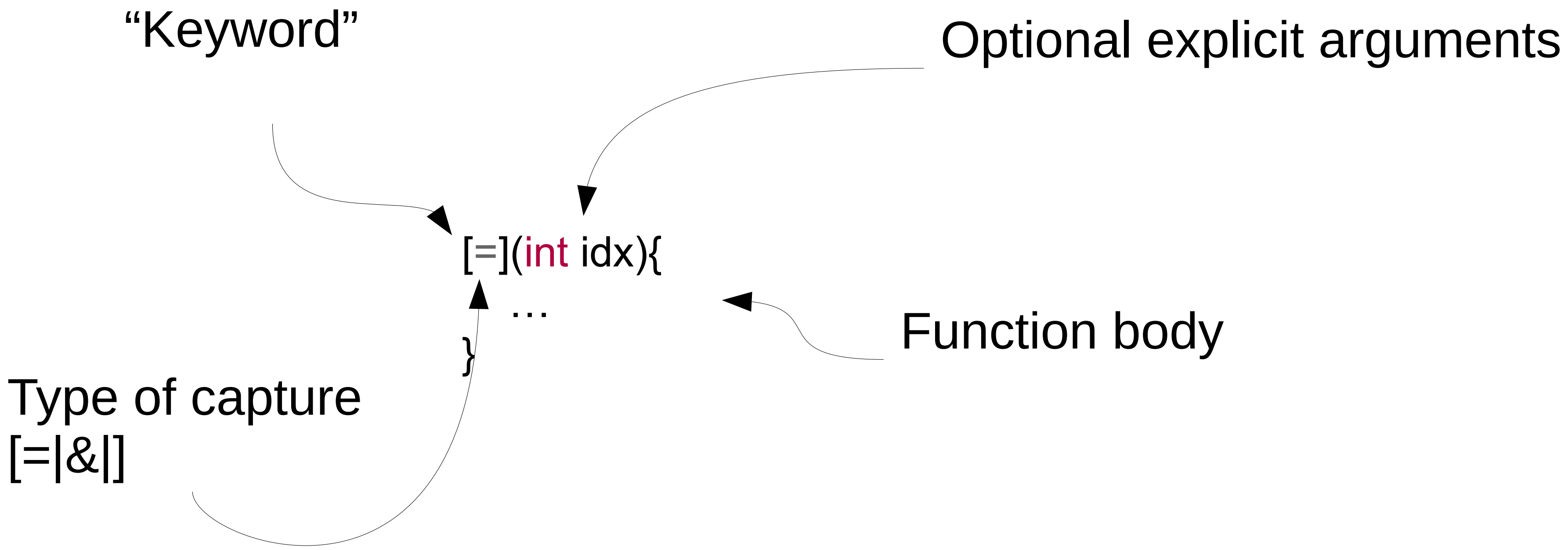
```
std::count_if(v.begin(), v.end(), [](const int& x){return (x >= 3 && x < 6);});
```

LAMBDAS

```
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);
```

LAMBDAS

Lambdas are anonymous functions that can capture variables.



```
[=](int idx){
   …
}
```

STD::TRANSFORM + LAMBDAS

```
#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

09-pSTL/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

09-pSTL/exercises/tasks/for_each

Compile with make.

Exercise

09-pSTL/exercises/tasks/thrust_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

09-pSTL/exercises/tasks/jacobi

Compile with make.

REFERENCES

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