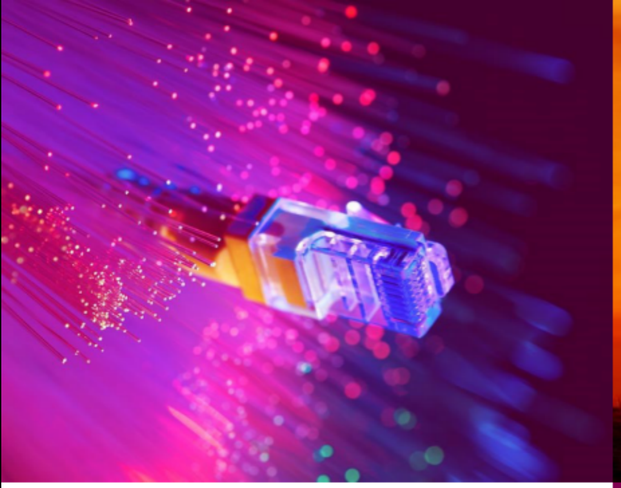

C++ Refresher

High-Level GPU Programming

2024-02

CSC Training


CSC – Finnish expertise in ICT for research, education and public administration

C++ Refresher (for those familiar with C)

Anatomy of a SYCL code

```
#include <sycl/sycl.hpp>
using namespace sycl;

template <typename T>
void axpy(queue &q, const T &a, const std::vector<T> &x, std::vector<T> &y) {
  range<1> N{x.size()};
  buffer x_buf(x.data(), N);
  buffer y_buf(y.data(), N);

  q.submit([&](handler &h) {
    auto x = x_buf.template get_access<access::mode::read>(h);         // accessor x(x_buf, h, read_only);
    auto y = y_buf.template get_access<access::mode::read_write>(h);   // accessor y(y_buf, h, read_write);

    h.parallel_for(N, [=](id<1> i) {
      y[i] += a * x[i];
    });
  });
  q.wait_and_throw();
}
```

- SYCL and Kokkos are modern C++ with classes, templates, lambdas, ...

Namespaces

- Namespace is a way of organizing variables, functions, classes, etc.

```
// Fully qualified name
sycl::queue q{};

// Using names from the namespace
using namespace sycl;
queue q{};
```

Templates

- Templates allow writing generic functions and classes

```
template <typename T>
T max(T a, T b) {
  return (a > b) ? a : b;
}

int a = 1, b = 2;
int c = max(a, b);

double x = 3.4, y = 5.6;
double z = max(x, y);
```

Pointers and references

- Pointer: Memory address of another variable (as in C)

```
void foo1(int *a) { *a = 42; }

int *x = new int[1];
foo1(x);
std::cout << *x << std::endl;
delete[] x;
```

- Reference: Alias of another variable

```
void foo2(int &a) { a = 42; }

int y;
int &z = y;
foo2(z);
std::cout << y << std::endl;
```

auto

- `auto` can be used in variable declaration if the compiler can deduce the type during compilation

```
auto a = 5;

auto queue_event = queue.submit([&](handler& h) {...});
```

Lambdas

- Anonymous function objects
- Syntax: `[ captures ] (parameters) -> return-type { body }`

```
int a = 1;
auto add = [=](int b) -> int { return a + b; };
int sum = add(2);  // 3

auto set = [&](int b) { a = b; };
set(5);  // a = 5

sum = add(2); // 3 or 7?
```

Classes

- Composite data type grouping variables and functions

```
template <typename T>
class Particle {
private:
    T x, y;

public:
    Particle(T x, T y) {
        this->x = x;
        this->y = y;
    }

    void move(T dx, T dy) {
        x += dx;
        y += dy;
    }

    void print() {
        std::cout << x << " " << y << std::endl;
    }
};
```

```
Particle<double> p{1.2, 3.4};
p.print();
p.move(5.6, 7.8);
p.print();
```

Functors

- Objects that behave as functions

```
class Adder {
private:
    const int constant;
public:
    Adder(int c) : constant{c} {}
    int operator()(int a) { return constant + a; }
};

Adder add{5};
int sum = add(2);
std::cout << "The sum is: " << sum << std::endl;
```

Error Handling

- Errors are handled via C++ exceptions

```
int main() {
  int x, y;
  std::cout << "Enter two numbers: ";
  std::cin >> x >> y;

  try {
    if (y == 0) throw "Division by zero error";
    std::cout << "x / y = " << x / y << std::endl;
  } catch (const char* msg) {
    std::cerr << "Error: " << msg << std::endl;
  }
  return 0;
}
```

Summary

- SYCL and Kokkos are modern C++ aiming towards generic parallel programming
- Classes, templates, lambdas, …
- Reusable, expressive, and efficient code