C++11/14 features relevant in GPGPU APIs

GPU Day 2015

The Future of Many-Core Computing in Science

Jeffrey Kelling

20th May 2015

C++11

- increased expressiveness of language
  - shorter code, less error-prone
- more high-level compile-time abstractions
  - templates ..., move-semantics, ...

- adress today's common tasks in STL
  - thread support, regular exressions, ...

C++11 feels like a new language: The pieces just fit together better than they used to and I find a higher-level style of programming more natural than before and as efficient as ever.

—B. Stroustrup
http://stroustrup.com/C++11FAQ.html

Expressiveness? Abstractions?

```
__device__ void count(int *count, char *data, size_t n, char value)
{
    for (int i = blockDim.x * blockIdx.x + threadIdx.x;
         i < n;
         i += gridDim.x * blockDim.x) // verbose
    {
        if (data[i] == value) atomicAdd(count, 1);
    }
}
```

- common pattern in GPGPU
- – complex iteration promotes bugs

[0] CUDA 7 talk of Mark Harris at GTC 2015, S5820

Expressiveness? Abstractions?

```
__device__ void count(int *count, char *data, size_t n, char value)
{
    const grid_stride_range range(0, n);
    for (grid_stride_range::const_iterator i = range.begin();
         i != range.end(); ++i) // C++03
    {
        if (data[*i] == value) atomicAdd(count, 1);
    }
}
```

+ most mistakes will be cought by compiler
− still repetitive

[0]CUDA 7 talk of Mark Harris at GTC 2015, S5820

Expressiveness? Abstractions?

```
__device__ void count(int *count, char *data, size_t n, char value)
{
    for (auto i : grid_stride_range(0, n) ) // C++11
    {
        if (data[i] == value) atomicAdd(count, 1);
    }
}
```

- auto —infers type from initializer
- for(:) —range-based for-loop

[0] CUDA 7 talk of Mark Harris at GTC 2015, S5820

Expressiveness? Abstractions?

```
// generic for type and predicate
template <typename T, typename Predicate>
__device__ void count_if(int *count, T *data, size_t n, Predicate p)
{
    for (auto i : grid_stride_range(0, n) ) // C++11
    {
        if (p(data[i])) atomicAdd(count, 1);
    }
}
```

- we want to write our algorithms *once*

[0] CUDA 7 talk of Mark Harris at GTC 2015, S5820

Type-Inference

```
// state type only once
auto object = new SomeTemplateClass<...> (...);
// same type as object
decltype(object) object2 = 0;

// leave out template args and nested classes
std::list<SomeTemplateClass<...>> list;
auto i = list.begin();

// beware:
auto copy = object->getRefToMember();
auto& ref = object->getRefToMember();

// deduction of return type
auto answer() { return 42; }
```

Lambda

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

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

- Unnamed function object
- captures variables from scope
  - [&] by reference
  - [=] by value

```
auto lambda = [&letters](char c) -> bool { /* ... */ };
auto lambda = [=](auto c /* C++14, generic lambda */ ) { /* ... */ };
```

[0] CUDA 7 talk of Mark Harris at GTC 2015, S5820

rvalue References and Move Semantics

- rvalue references: move instead of copy

```
class A {
    A (A&&);                 // move constructor
    A& operator= (A&&);  // move assignment
    // ...
};

#include <utility>

template<class T>
void swap(T& a, T& b)
{
    T tmp = std::move(a); // move constructor
    a = std::move(b);       // move assignment
    b = std::move(tmp);
}
```

- STL containers have move constuctors and assigments

[0] http://stroustrup.com/C++11FAQ.html#rval

Variadic Templates

```
// python-style print
template<class T>
void print(T arg)
{
    std::cout << arg << '\n';
}

template<class T, class... Types>
void print(T arg, Types... rest)
{
    std::cout << arg << '␣';
    print(rest... );
}
```

- `parameter pack`[1] may occur in many places
- sizeof ... paramPack gives number of elements

[1] http://en.cppreference.com/w/cpp/language/parameter_pack

Constructor Inheritance

```
struct Base {
    Base();
    Base(int);
    Base(int, int);

    virtual f();
    //...
};

struct A : public Base {
    // inherit constructors
    using Base::Base;
    A() : A(5) {} // delegate constructor

    virtual f();
    //...
};

struct B : public Base {
    // inherit constructors
    using Base::Base;
    B(int, int); // hide Base(int, int)
    B() : Base() {}

    virtual f();
    //...
};
```

Variadic Templates—Example

```
#include <memory>
#include <utility>

// generic factory method
template<class... CArgTypes>
auto factory(char subClass, CArgTypes... cArgs) {
    switch(subClass) {
        case 'A':
            return std::unique_ptr<Base>(
                new A(std::forward<CArgTypes>(cArgs)...));
        case 'B':
            return std::unique_ptr<Base>(
                new B(std::forward<CArgTypes>(cArgs)...));
        default:
            return std::make_unique<Base>(
                std::forward<CArgTypes>(cArgs)...);
    }
}

int main() {
  Base* a  = factory('A', 23);
  Base* b  = factory('B', 23, 42);
  Base* b2 = factory('B');

  //...
}
```

STL Thread Support

- thread, mutex, future, async ...

```
#include <thread>
#include <list>
#include <random>

int main()
{
    std::list<double> list {1,2,3,4,5,6,7,8,9}; // initializer list

    std::random_device rd;
    std::mt19937 gen(rd());
    std::discrete_distribution<> d({1, 1});

    auto thread = std::thread(
        // function to call, disambiguate
        [&list](auto pred){list.sort(pred);},
        // arguments (here: sorting predicate)
        [&d,&gen](double a, double b) {return bool(d(gen));}
    );

    // done something else ...
    // ... but do not touch list

    thread.join();

    return 0;
}
```

```
g++ -std=c++14 main.cpp -lpthread
```

Things I left out

- **Language**
  - uniform/value initialization, alignment, final, noexcept, ...
- **STL**
  - `std::bind`—bind parameters of function in new callable
  - tuple—generalization of `std::pair`
  - improved smart pointers
  - containers: `array`, `forward_list`, ...
  - math, random numbers
  - time measurement `<crono>`
  - atomics, regular expressions
  - ...

[2] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4352.html

[3] http://en.cppreference.com/w/cpp/language/fold

Things I left out

- **Language**
  - uniform/value initialization, alignment, final, noexcept, ...
- **STL**
  - `std::bind`—bind parameters of function in new callable
  - tuple—generalization of `std::pair`
  - improved smart pointers
  - containers: `array`, `forward_list`, ...
  - math, random numbers
  - time measurement `<crono>`
  - atomics, regular expressions
  - ...
- **C++17**
  - concepts
  - parallel algorithms[2]
  - fold expressions[3], ...

[2] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4352.html

[3] http://en.cppreference.com/w/cpp/language/fold

The End

J.Kelling@HZDR.de

NVIDIA GPU CENTER OF EXCELLENCE

Thank You.