Easy::Jit

Just-In-Time compilation for C++ codes

Serge Guelton (two presentations from)
Juan Manuel Martinez Caamaño (me)

Introduction

✓ Compiler-assisted library for runtime code generation

✗ An omniscient virtual machine

✗ Building blocks for a Just-in-Time compiler

Easy::Jit: by example

```cpp
static void apply_filter(const char *mask, unsigned mask_size, unsigned mask_area,
                         cv::Mat &image, cv::Mat *&out) {


    kernel(mask, mask_size, mask_area,
           image.ptr(0,0), out->ptr(0,0),
           image.rows, image.cols, image.channels());
}
```

Easy::Jit: by example

```cpp
#include <easy/jit.h>


static void apply_filter(const char *mask, unsigned mask_size, unsigned mask_area,
                         cv::Mat &image, cv::Mat *&out) {
   using namespace std::placeholder;


   auto callme = easy::jit(kernel, mask, mask_size, mask_area,
                           _1, _2, image.rows, image.cols, image.channels());

    callme(image.ptr(0,0), out->ptr(0,0));
}
```
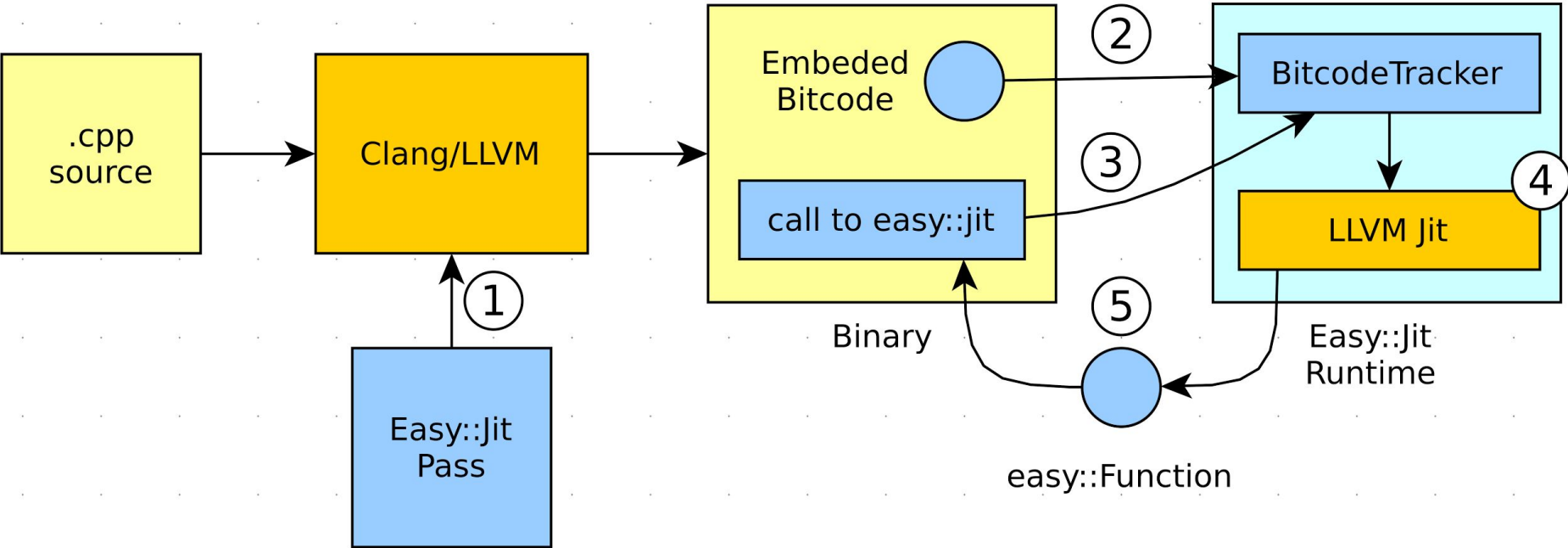
Easy::Jit: by example
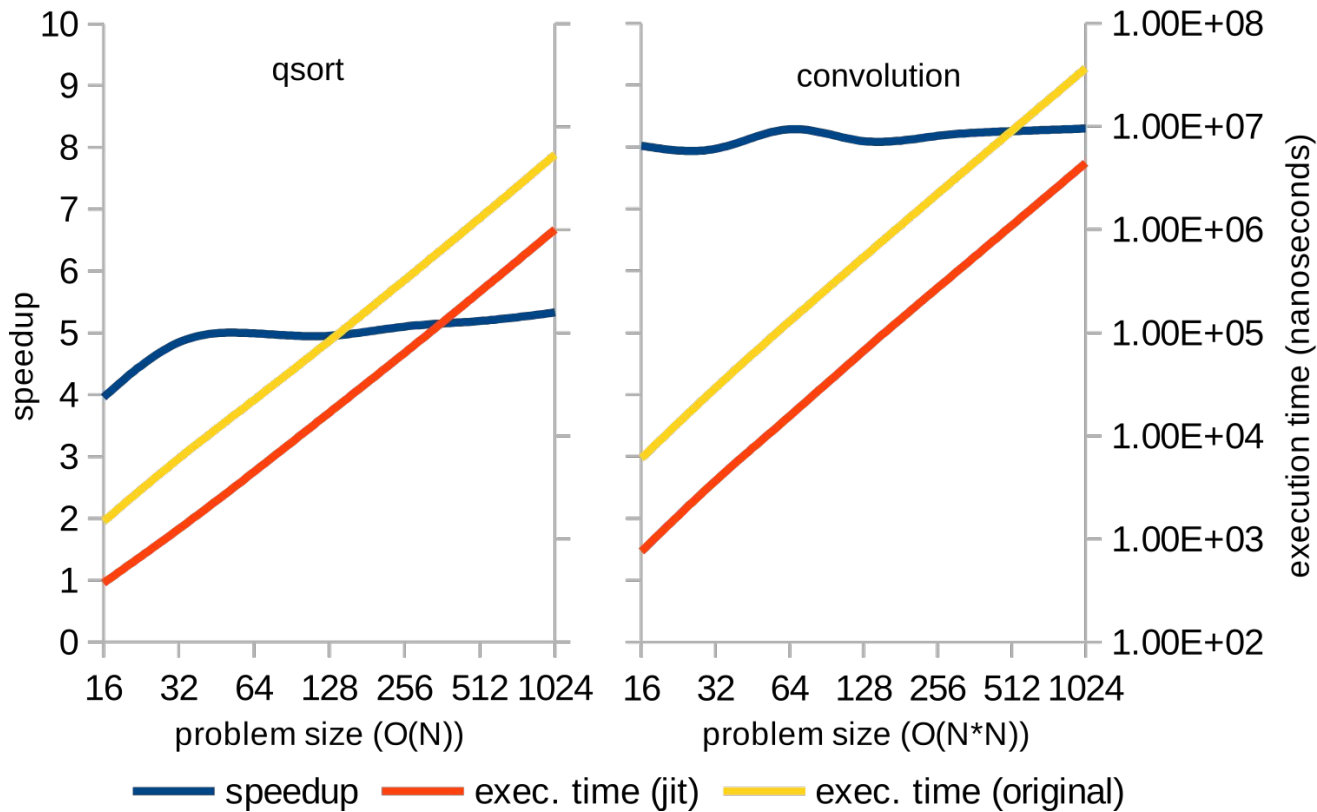
```cpp
#include <easy/code_cache.h>


static void apply_filter(const char *mask, unsigned mask_size, unsigned mask_area,
                         cv::Mat &image, cv::Mat *&out) {
    using namespace std::placeholder;
    static easy::Cache cache;
    auto const& callme = cache.jit(kernel, mask, mask_size, mask_area,
                         _1, _2, image.rows, image.cols, image.channels());
     callme(image.ptr(0,0), out->ptr(0,0));
}
```

Easy::Jit: Internals

Easy::Jit: The numbers

Easy::Jit: Roadmap

➔ CloneModule to another LLVMContext
   ◆ More advanced threading / caching techniques
   ◆ Specialising for functions generated dynamically
➔ Serialization/Deserialization of compiled code
   ◆ Persistent caching
➔ Member functions and function objects
➔ Immutable parameters

```
void eval(AST* ast, int variables[]);
...
auto program = easy::jit(eval, ast, _1);
program(var_values)
```

???

Merci **Quarkslab** :)

github.com/jmmartinez/easy-just-in-time