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
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

#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

1. .cpp source
2. Embedded Bitcode
3. Call to easy::jit
4. BitcodeTracker
5. LLVM Jit

Easy::Jit Pass

Binary

easy::Function

Easy::Jit Runtime
Easy::Jit: The numbers

The graphs show the comparison of speedup and execution time between qsort and convolution algorithms. The x-axis represents the problem size, which is either O(N) or O(N*N), and the y-axis represents the speedup or execution time (nanoseconds). The graphs illustrate the performance improvement of qsort and convolution as the problem size increases.
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

```cpp
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