Adding LLVM JIT facility to your program.

Syoyo Fujita

Agenda

Motivation

How to do it

Conclusion

Motivation

Compile the code online

Fast code execution

How to do it

1) Read or construct LLVM module

2) Apply optimizer pass(if want)

3) Compile to native code

3) Get a function pointer to compiled code

4) Call a function pointer to evaluate

ExecutionEngine

A key module to use JIT facility with LLVM.

Can compile a function in the module into the native code.

Provides a interface to call JIT-ted function.

Includes

```
#include "llvm/Module.h"
#include "llvm/Constants.h"
#include "llvm/DerivedTypes.h"
#include "llvm/Instructions.h"
#include "llvm/ModuleProvider.h"
#include "llvm/ExecutionEngine/JIT.h"
#include "llvm/ExecutionEngine/Interpreter.h"
#include "llvm/ExecutionEngine/GenericValue.h"
#include <iostream>
using namespace llvm;
```

```
// Now we create the JIT.
ExistingModuleProvider* MP = new ExistingModuleProvider(M);
ExecutionEngine* EE = ExecutionEngine::create(MP, false);
```

Create a ExecutinEngine for module M.

```
Function *FooF =
  cast<Function>(M->getOrInsertFunction(
    "foo", Type::Int32Ty, (Type *)0));
```

Get a (LLVM) function to be JIT-ted from module M.

JIT & exec

```
// Call the function with argument n:
std::vector<GenericValue> Args(1);
Args[0].IntVal = APInt(32, n);

GenericValue GV = EE->runFunction(FooF, Args);
```

Compile a function and execute it.

Provides function argument through GenericValue array.

JIT only

```
void *FPtr = TheExecutionEngine->getPointerToFunction(FooF);
```

JIT the function, returning a function pointer.

Limitation

You need to know function signature a priori to call the function.

LLVM provides a facility to get a function signature.

LLVM can JIT a program per function.

Cannot per more fine unit(e.g. BasicBlock)

References

http://llvm.org/docs/tutorial/

$(LLVM)/examples/HowToUseJIT