LLVM Just-in-Time Compiler

Rong Yuyang

ShanghaiTech University
School of Information Science and Technology(SIST)

rongyy@shanghaitech.edu.cn

July 4, 2018

Overview

What is Just in Time

- Program stored in Bytecode;
- Furture compiled right before it's called;
- Compilation happens *Just in Time*.

Two Engines in LLVM

- JIT(Old one, to be removed after llvm 3.5);
- MCJIT(New one utlizing MC).

JIT

- Function based;
- Compile one function at a time;
- Compilation happens when function pointer is generated or called(lazy compilation).

How to use JIT

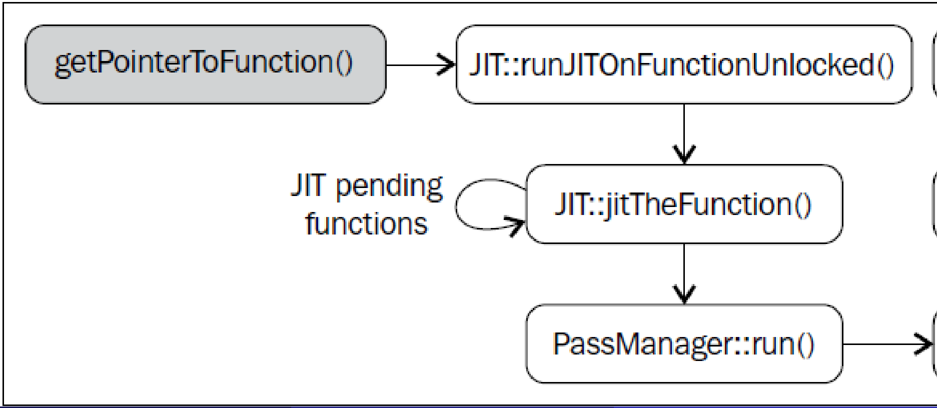
- Load *.bc* file;
- Construct a parser on the file;
- Based on the parser construct an execution engine;
- Load one specific function using parser and throw it to the executuon engine;
- Woola! You got a function pointer pointing to a newly compiled function;
- Call the function.
- Release Memory and shutdown llvm

How to use JIT cont.

```
MemoryBuffer::getFile("./sum.bc", Buffer);
Module *M = ParseBitcodeFile(Buffer.get(), Context,
    &ErrorMessage);
OwningPtr<ExecutionEngine> EE(EngineBuilder(M).create());
Function *SumFn = M->getFunction("sum");
int (*Sum)(int, int) = (int (*)(int, int))
    EE->getPointerToFunction(SumFn);
int res = Sum(4,5);
EE->freeMachineCodeForFunction(SumFn);
llvm_shutdown();
```

How JIT works

- Memory Manager: *JITMemoryManager*;
- Code emitter<*Target*>*CodeEmitter*;
- Target Information(Classes that targets on different platforms *TargetJITInfo*)



Use of generic value

- Use a *vector<GenericValue>* as parameters.
- Call *runFunction* without getting a function pointer.

```
vector<GenericValue> FnArgs(2);
FnArgs[0].IntVal = APInt(32,4);
FnArgs[1].IntVal = APInt(32,5);
GenericValue Res = EE->runFunction(SumFn, FnArgs);
```

Difference

- Module based. Compilation happens for one module at a time.
- Utilizes MC framework we talked in the last Lecture by Zhiqiang.

State definition

- Added: Not compiled but already added to the engine;
- Loaded: Compiled but not ready for execution;
- Finalized: functions executable.

Interpreter, JIT and Ahead of Time(AOT)

- Interpreter: Shine :)
- AOT: C, C++, Rust.
- JIT(Somewhere in between): Java, C#, etc.

Pros and Cons

- Pro: Better performance than interpreter.
- Pro: Utilizes compile time optimization while using better run-time optimization than AOT
- Pro(or Con): Potability. Code once, use(or debug) everywhere.
- Con: Translating during run-time takes time(Slow).
- Con: Are Readable, Writable and Executable memory pages safe?

Reference

- Getting started with LLVM core libraries.
- https://softwareengineering.stackexchange.com/questions/246094/understanding-the-differences-traditional-interpreter-jit-compiler-jit-interp
- https://www.whizlabs.com/blog/what-is-just-in-time-compiler-difference-between-compiler-and-interpreter/
- https://stackoverflow.com/questions/95635/what-does-a-just-in-time-jit-compiler-do
- https://en.wikipedia.org/wiki/Just-in-time_compilation