A gentle introduction to LLVM ORC JIT

樊其轩

Prelude: A simple JIT implementation

1. Load code in memory
2. Compile into executable object in Read-Write memory
3. Mark or Copy into memory executable and Read-Only
4. Go ahead and run

http://nickdesaulniers.github.io/blog/2013/04/03/basic-jit/

map/remaphttps://kazlauskas.me/entries/fast-page-maps-for-jit

On request compilation(ORC)

- Eager compilation (Compilation on lookup)
- Lazy compilation → reexport & indirection
- Speculative compilation
- Concurrent compilation
- Remote execution & debugging: jit thread → llvm-jit-executor ← debugger
- Mimic run-time environment (ehframe, (de)initializors of static class, dlopen, …)
- Mimic a usual linker (link order, static archive, harness, weak/strong symbols, etc)
- Link backend: Jitlink or RuntimeDyld
- Handy API: LLJIT LazyLLJIT, ref. Kaleidoscope

Use cases

- GDB expression evaluation
- Interpreted Languages: clasp [1], Julia, llvm IR interpreter(lli)
- LLVM pipe (Mesa Gallium software rendering backend, headless testing e.g. xvbf)
- Other interesting usage: C++ as a scripting language (see 2. below)

1. Common lisp implemented with llvm that supports C++ interfacing https://github.com/clasp-developers/clasp
2. Handling all Facebook requests with JITed C++ code

Abstraction & Facilities

- Layers: ObjectLinkingLayer, IRLayer
- JITDylib
- MaterializationUnit & MaterializationResponsiblity
- ResourceTracker & ResourceManager
- ExecutionSession

- DefinitionGenerator
- PlatformSupport & xxPlatform
- ExecutorProcessControl, EPCxx, TargetProcess/*, ...
- IndirectionStubManager, LazyCallThroughManager

Layers

ORC (Lazy)LLJIT

- (Compile On-demand layer)
- IRTransformLayer
- IRCompileLayer
- ObjectTransformLayer
- ObjectLinkingLayer
- JITLink or RuntimeDyld

old MCJIT

- IR
- MC
- Object
- RuntimeDyld

  - void emit(std::unique_ptr<MaterializationResponsibility> R, ThreadSafeModule TSM)
  - Error add(JITDylib &JD, ThreadSafeModule TSM)

JITDylib

Resembles a usual .so library, but don’t require all its content compiled and available at once. JITDylibs are searched and linked with DFS order

Contains:

- SymbolTable, {Unmaterized, Materializing}Info, linkorder, ResourceTracker

Interface:

- addToLinkOrder(JITDylib &JD, JITDylibLookupFlags JDLookupFlags
- Error define(std::unique_ptr<MaterializationUnitType> &&MU, ResourceTrackerSP RT = nullptr)
- Error remove(const SymbolNameSet &Names)
- Error resolve(MaterializationResponsibility &MR, const SymbolMap &Resolved)
- Error emit(MaterializationResponsibility &MR, const SymbolFlagsMap &Emitted)

Materialization{Unit,Responsiblity}

- xxUnit

  A set of symbols that can be managed and emitted together

  - void materialize(std::unique_ptr<MaterializationResponsibility> R)
  - const SymbolFlagsMap &getSymbols()

- xxResponsiblity

  Get transferred through layers and link phases.

  Tracks Unit materializing

  >> proxies handling of failure and completion of symbol emission&resolution (essential pattern for concurrent compilation).

ExecutionSession

Represent a running JIT program session

contains:

- JITDylibs, EPC, Platform, Mapping of RT & symbols & MR, ResourceManagers, OutstandingMUs

functions:

- lookup()
- OL_xx(), IL_xx()
- Expected<JITDylib &> createJITDylib(std::string Name);
- std::unique_ptr<MaterializationResponsibility> createMaterializationResponsibility(ResourceTracker &RT, SymbolFlagsMap Symbols, SymbolStringPtr InitSymbol)

EPC(ie ExecutorProcessControl)

Blurs the boundary of JIT compiler & executor

- SelfExecutorProcessControl
- SimpleRemoteEPC

symbol string pool, memory manager, remote execution(talk to server), SPS(simple packing serializer)

Other facilities

- PlatformSupport & xxPlatform
- DefinitionGenerator
- Ehframe, Ctor Dtor
- Debug support
- Target executor

Code exploration 1

The Simplest path of a LLVM IR’s life in Orc

Code exploration 2

How are (de)initializors supported?

Code exploration 3

{AbsoluteSymbol, Reexport}MaterializationUnit

{Linkgraph, Object, IR}MaterializationUnit

Reference and more materials

- 2016 ORC - LLVM's Next Generation of JIT API
- 2018 Updating ORC JIT for Concurrency
- 2021 ORCv2 - LLVM JIT APIs Deep Dive
- LLVM discord server #JIT channel

Thank you !