Case study: developing an analysis and instrumentation tool based on LLVM: PARCOACH

LLVM @ FOSDEM 2023 - February 4th

Philippe Virouleau

Outline

Inria

01

Introduction and context

Inria

Why?

- Provide a feedback, lay down what I wish I knew before
- Encountered similar issues in various out-of-tree projects
- The talk is not so much about the tools themselves but rather about the approach.

For whom?

Anyone (about to be) involved in an out-of-tree LLVM tool/plugin.

Disclaimer: this is my own take on this topic, if you have alternatives or better ways of dealing with what I will describe, please do let me know :)

PARCOACH[1]

Analysis and instrumentation tool for HPC.
Detects incorrect usage of OpenMP/MPI's APIs (data race, deadlock, ...).

- Devs: Interns, PhD students, researchers
- Users: scientific applications devs, students
- Started with LLVM 3.7, now LLVM 15
- No dedicated LLVM engineer until recently

1: `https://gitlab.inria.fr/parcoach/parcoach`

Commercial compiler

Commercial LLVM based obfuscator.

- Devs: LLVM/C++ engineers
- Users: clients

Student LLVM exercises[2]

Introduction to code transformation with LLVM (15)

- Devs: Juan, me
- Users: students

2: `https://github.com/viroulep/master-csi-public`

02

Keeping up with LLVM

Inria

Naive/manual approaches

Either forced:

- With LLVM < 3.5: manual compilation, using `llvm-config`
- Manual `add_library` + `target_link_libraries`

Or based on user's experience

- "CMake integration" but with hardcoded values

Using CMake integration

Simplifies build options:

- What lib for which components?
- What if I want to link dynamically? Statically?

Dedicated macros: `add_llvm_library`, `add_llvm_pass_plugin`, `add_llvm_tool`

CMake - example code

```
1  # User can pass -DLLVM_DIR to help CMake
2  find_package(LLVM 15 REQUIRED CONFIG)
3  list(APPEND CMAKE_MODULE_PATH "${LLVM_CMAKE_DIR}")
4  include(AddLLVM)
5  # Make the LLVM definitions globally available.
6  add_definitions(${LLVM_DEFINITIONS})
7  include_directories(${LLVM_INCLUDE_DIRS})
8
9  set(LLVM_LINK_COMPONENTS Core Support Passes ...)
10 # Or STATIC
11 add_llvm_library(mylib LibSource.cpp SHARED)
12
13 # Maybe pass DISABLE_LLVM_LINK_LLVM_DYLIB
14 add_llvm_tool(mytool Source.cpp)
15 target_link_libraries(mytool mylib)
16
17 # ... Somewhere else (or unset ${LLVM_LINK_COMPONENTS})
18 add_llvm_pass_plugin(myplugin PluginSource.cpp)
```

LLVM's cmake sets libs and targets based on the build you want.

(more on this in the packaging section)

Useful examples: `llvm/examples/Bye`, `llvm-tutor`

Inria

Familiarity with C++/LLVM

- (New) contributors may not be comfortable
- Code taken from "old" snippets

Often seen idioms:

```
for (auto ItBB = F.begin();
     ItBB != F.end(); ++ItBB) {
  for (auto It = BB->begin();
       It != BB->end(); ++It) {
    Instruction &I = *It;
    // ...
  }
}
```

Alternatives:

```
for (auto &BB : F) {
  for (auto &I : BB) {
    // ...
  }
}
```

```
for (auto &I : instructions(F)) {
  // ...
}
```


Inria

Familiarity with C++/LLVM

- (New) contributors may not be comfortable
- Code taken from "old" snippets

```
for (auto &I : instructions(F)) {
  if (!isa<CallInst>(I)) {
    continue;
  }
  CallInst &CI = cast<CallInst>(I);
  // ...
}
```

```
for (auto &I : instructions(F)) {
  if (CallInst *CI =
        dyn_cast<CallInst>(&I)) {
    // ...
  }
}
```

```
auto IsCI = [](Instruction &I) {
  return isa<CallInst>(I);
};
for (auto &I : make_filter_range(
      instructions(F), IsCI)) {
  CallInst &CI = cast<CallInst>(I);
  // ...
}
```

First approach is to use "known" data types:

```
void foo(std::map<Instruction *, int> &Input, int X) {
  bar(Input); // Do something with Input
  auto Found = std::find_if(Input.begin(), Input.end(),
    [](auto const &Entry) { return Entry.second == X; });
  if (Found != Input.end()) {
    Instruction *I = Found->first;
    int Val = Found->second;
    // Do something with I and Val.
  }
}
```

What if the Value is modified (deleted, RAUW-ed)?

```
void foo(ValueMap<Instruction *, int> &Input, int X) {
  bar(Input); // Do something with Input
  auto Found = llvm::find_if(Input,
    [](auto const &Entry) { return Entry.second == X; });
  if (Found != Input.end()) {
    auto [I, Val] = *FoundI;
    // Do something with I and Val.
  }
}
```

Same goes for `*Vector`, `ArrayRef`, `StringRef`, and all of `STLExtras`...

Is it being picky?

- Depends on who is contributing
- Accumulation of small details matters
- Makes code more readable (= easier for new contributors)

Some ideas

- Code reviews (obvious in some context, hard to do in others). (eg: research in areas where compilers are "just" a tool).
- Read LLVM programmers manual.
- Read the code.

Common considerations

- API breaking.
- IR changes (eg: opaque pointers).
- May be time consuming (eg: PM migration).
- Dealing with deprecated elements.

Skipping versions makes it worse.

Supporting multiple LLVM versions

eg: any LLVM from 9 to 12.

Don't do it

Reminder: passes types

- Analysis (no IR change, cached, can be invalidated).
- Transformation (may change IR, can invalidate analyses).

Analyzing in transformations

Seen a lot of "all in one" passes, motivation for untangling them:

- Semantically different.
- Benefit from the caching system.
- Avoids passing structures around.

Obviously ease the PM migration.

Manual approach

- Manage timers (in different ways).
- Extra steps to get meaningful representations.
- Commented `llvm::errs()` everywhere.

LLVM structures

Timers through `TimeTraceScope`:

- One-line to create a named scoped timer
- Get a flame-graph as a bonus

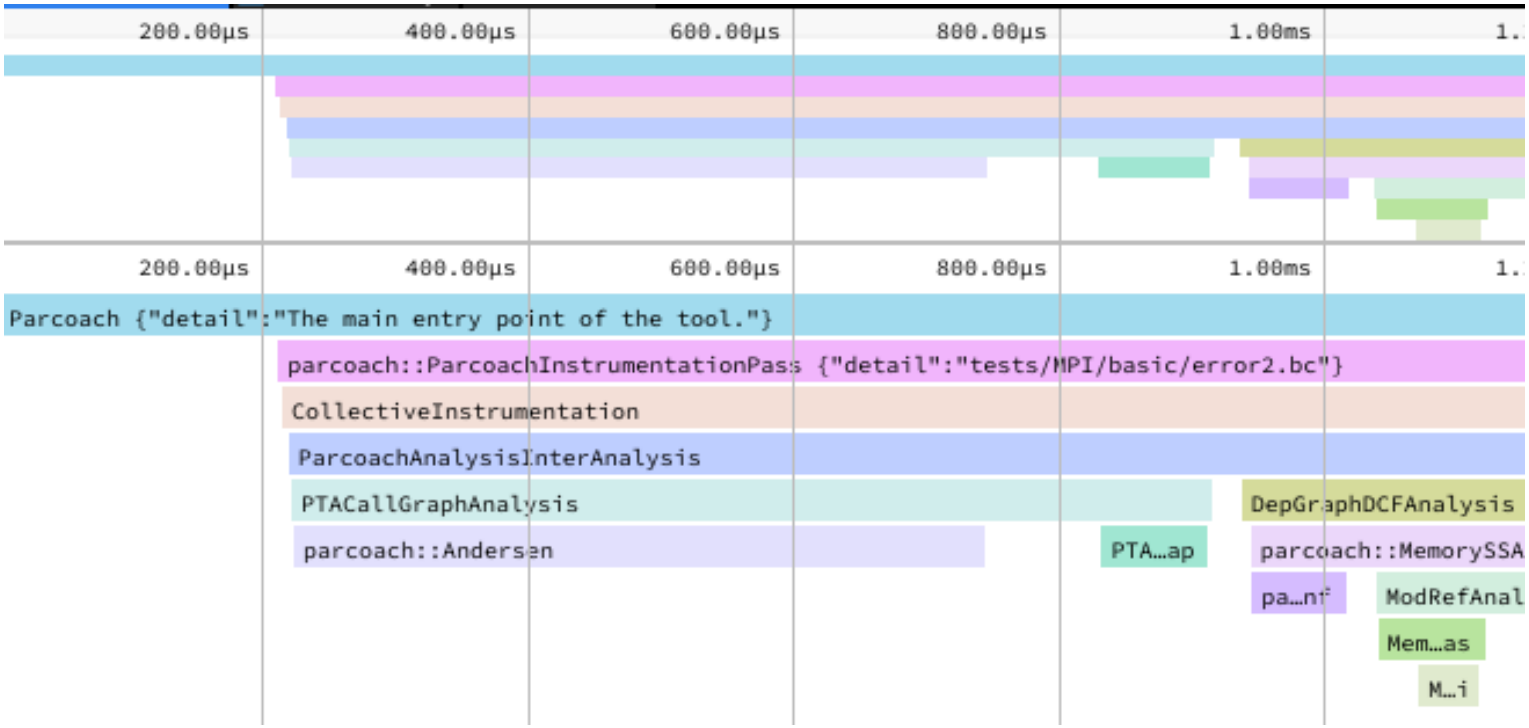
Debug system:

```
#define DEBUG_TYPE "mypass"
LLVM_DEBUG(dbgs() << "Some debug string");
```

Combined with `opt -debug-only="mypass"`.

Flamegraph example

Json imported in speedscope:



Make your life easy

- Invest in maintainance (if possible).
- Get inspiration from LLVM sources.
- Don't reinvent the wheel.

Code review is a great way to achieve that.

Keep the diff minimal

Upstream/use upstreamed passes.
Custom analyses is one of the main weaknesses of the project.

03

Usability

Inria

Make it easy to get started

- Make it clear what LLVM versions and features are needed. (lib/tools/utils)
- Good feedback from using docker (and clear CI). (it "just works", can code on the host)
- Assume not everyone knows LLVM (PhD students, interns).

Benefit from LLVM tooling

- Lit and FileCheck are great (need a release with tools/utils)
- Out-of-tree plugin/tool: makes sense to follow LLVM coding standards
  (base style for clang-format/clang-tidy)

Getting the tool

- Compile from source
- Figure out LLVM installation on their own
  (or compile it from source)

Usage

- Get IR (`clang/flang -emit-llvm`)
- `opt -load-pass-plugin=lib.so -passes=somename`

Is it nice enough? (for researchers, students)
Verification tool: running on each file is tedious, we need integration with autotools/cmake.

- Wrapper looking like "`parcoach clang -c a.c -o a.o`"
- Generate original object + generate tmp IR + run the tool

Docker

- Provide a (controlled) ideal setup
- Docker-compose to ease mounting some folder, or running on some file
- Ideal for students: code on the host, run in container
- Avoids the whole "how is LLVM packaged on everyone's computer?" point

Downside: not something you can do on shared clusters.

04

Dealing with packaging

Inria

DIY

- Ship a shared library.
- Depends on the installed opt.
  How is it compiled, what PM is enabled by default?

"Proper" package

- How is LLVM packaged for your target?
  Ubuntu's apt, LLVM's apt, Guix, Module file?

Full build (aka ship opt/[lib]LLVM)

- Useful if no known working LLVM version (eg: using LLVM 15 on custom RHEL 8.6)
- Ship all LLVM vs ship one single statically "small" tool

Set the options you want, LLVM's CMake handles it!

05

Conclusion

Inria

Takeaways

- Integration with LLVM has evolved (IMO in a good direction).
- Be prepared for maintainance.
- Keep the diff minimal or upstream your passes.
- Investing in CI is worth it (for devs and users).
- LLVM documentation (programmers' manual, doxygen) is a must, reading source code teaches a lot!

Questions, comments?