Can we democratize JIT compilers?

Make a wheel with the wheel

Haoran Xu
Stanford University

A little bit about me…

- Undergrad @ MIT, studied…
  - some math → found myself too stupid for math → gave up
  - some CS theory → found myself too stupid for theory → gave up
  - some Japanese → only thing I still remember
- Intern & full-time database dev @ SingleStore:
  - SQL & DB features: secondary index, computed column, user-level page cache…
- now PhD @ Stanford
  - JIT compiler research

LLVM: Reusable Compiler Infrastructure

- Numerous use cases:
  - Static compilers: C, Rust, Zig, WebAssembly, Fortran…
  - Database JITs: PostgreSQL, SingleStore, Hyper…
  - Language JITs: Julia, Numba…
- So why reinvent the wheel?

LLVM: not the silver bullet for JIT

- JIT is really a after-thought for LLVM
  - LLVM produces very good code, but the compilation itself is *very* slow
    - Goal of JIT: minimize compile + execution time, not execution only!
  - LLVM designed for static languages
    - Cannot support dynamic language optimizations
    - No dynamic language VM uses LLVM now (many failed attempts in the past)
  - Deployment issues
    - Huge library, no backward/forward compat

Can we have a better reusable infrastructure for JIT compilers?

Copy-and-Patch

- How can we generate code without LLVM?
    - *And*, without reinventing LLVM?
- Idea: use LLVM to pre-generate machine code snippets (“stencils”)
    - that can be *configured* and stitched together at runtime.
- Stencil has holes:
    - Literals in the instructions (e.g., branch destination, immediate operands)
    - Registers in the instructions (unpublished)
- *Copy* stencil, *patch* holes → executable code

Paper: Copy-and-Patch Compilation, Haoran Xu & Fredrik Kjolstad, OOPSLA 2021
https://fredrikbk.com/publications/copy-and-patch.pdf

Copy-and-Patch

- Proof-of-concepts
  - SQL query compiler
    - ~1000x faster compilation than LLVM -O3, 24% slower code
  - WebAssembly compiler
    - 4.9x - 6.5x faster compilation than V8 Liftoff, also 39% - 63% faster code
- CPython 3.13 experimental Copy-and-Patch JIT (I'm not involved)
- MLIR research-prototype Copy-and-Patch backend (I'm not involved)
- Backend for a research graph database (I'm not involved)

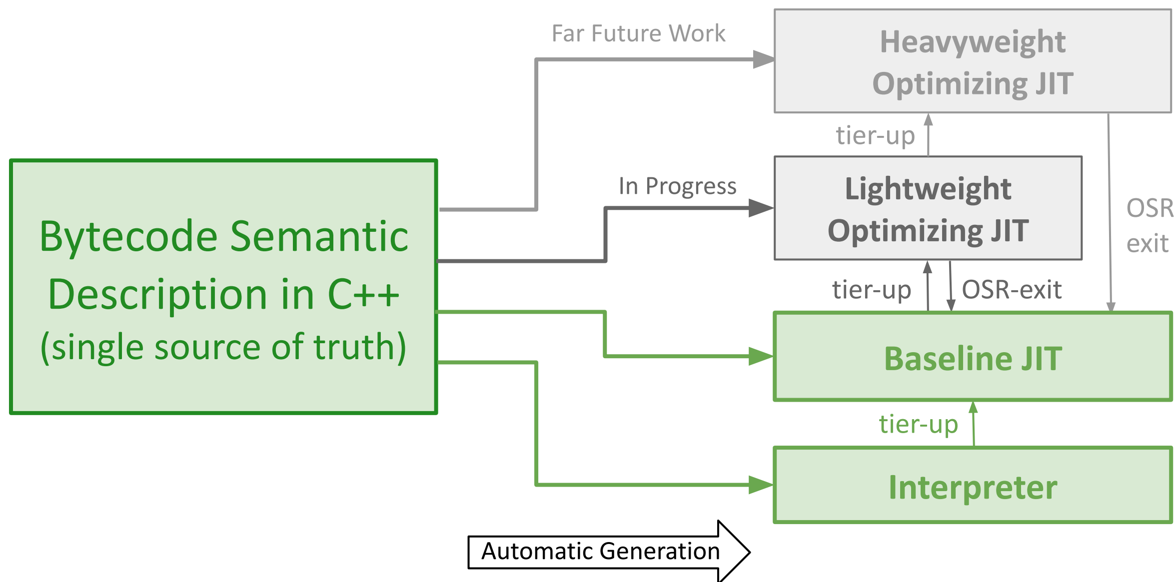
Paper: Copy-and-Patch Compilation, Haoran Xu & Fredrik Kjolstad, OOPSLA 2021
https://fredrikbk.com/publications/copy-and-patch.pdf

Copy-and-Patch

- Makes the very complex problem of writing a JIT much easier
  - Nevertheless, many hacks & rough edges, not very beautiful
- Solves the problem of code generation
  - But doesn’t solve the problem of higher-level optimizations
- But can we push one step further?

Deegen

- Ultimate goal: a reusable infrastructure for dynamic languages



Paper: Deegen: A JIT-Capable VM Generator for Dynamic Languages
Haoran Xu, Fredrik Kjolstad
https://arxiv.org/pdf/2411.11469

LuaJIT Remake

- Standard-compliant Lua 5.1 VM
- Interpreter: 179% faster than PUC Lua, 31% faster than LuaJIT interpreter
- Baseline JIT: 360% faster than PUC Lua, 33% slower than LuaJIT
  - Note that baseline JIT is not designed to compete with optimizing JIT on throughput!
  - We are working on generating optimizing JIT (to be covered shortly)

Paper: Deegen: A JIT-Capable VM Generator for Dynamic Languages
Haoran Xu, Fredrik Kjolstad
https://arxiv.org/pdf/2411.11469

Baseline JIT: Copy-and-Patch++

- Copy-and-Patch with dynamic-language-specific optimizations
  - Polymorphic inline caching
  - Hot-cold code splitting
  - Type-based optimizations
- See paper for more info

Paper: Deegen: A JIT-Capable VM Generator for Dynamic Languages
Haoran Xu, Fredrik Kjolstad
https://arxiv.org/pdf/2411.11469

Optimizing JIT: Copy-and-Patch[‡] + higher-level opts

- No publication or blog post yet, but code is available (google LuaJIT-Remake!)
- Copy-and-Patch with *truly generic register allocation*
- Generated IR definition:
  - Yes, we generate the *definition* of an IR!
- Frontend: bytecode → IR
- Speculative inlining
- Type speculation
- OSR exit map generation
- All work listed above are done, but many more TODO ideas on the table…

Thanks for your attention!

- Questions :)