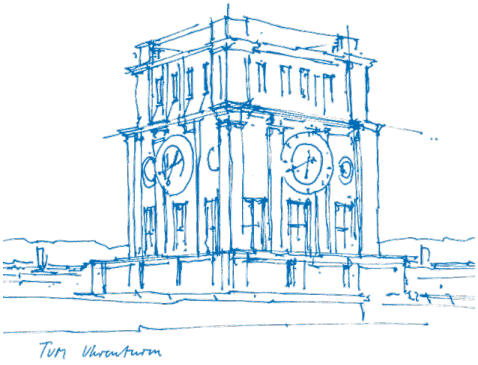
Compact Unwind Information for ELF

Alexis Engelke
engelke@tum.de

(with contributions from Fangrui Song)

Chair of Data Science and Engineering
Department of Computer Science
Technical University of Munich

EuroLLVM '26, Dublin, IE, 2026-04-15



Unwind Information

- Unwind info: meta-information describing stack frame layout
  - Describes: top of stack frame, location of callee-saved regs+ret addr, optionally: personality function and language-specific data (LSDA)
- Included in almost all executables/libraries:
  - For stack unwinding: C++ exceptions, Rust panics, debuggers
  - For stack tracing: profiling, crash stack traces
- Asynchronous unwind info: precise at every instruction

```
0x19970: CFA=RSP+8: RIP=[CFA-8]
0x19975: CFA=RSP+16: RBP=[CFA-16], RIP=[CFA-8]
0x19978: CFA=RBP+16: RBP=[CFA-16], RIP=[CFA-8]
0x1997e: CFA=RBP+16: RBX=[CFA-40], RBP=[CFA-16], R14=[CFA-32], R15=[CFA-24], RIP=[CFA-8]
0x199e2: CFA=RSP+8: RBX=[CFA-40], RBP=[CFA-16], R14=[CFA-32], R15=[CFA-24], RIP=[CFA-8]
0x199e7: CFA=RBP+16: RBX=[CFA-40], RBP=[CFA-16], R14=[CFA-32], R15=[CFA-24], RIP=[CFA-8]
```

Unwind Information in ELF: DWARF

- Section `.eh_frame`:
  - One FDE per function
  - Encodes unwind info as DWARF bytecode program
  - Variable length, often 20–64 B
- Section `.eh_frame_hdr`:
  - Binary search table, 8 B entries
  - Maps function to FDE

```
DW_CFA_advance_loc: 5 to 0x19975
DW_CFA_def_cfa_offset: +16
DW_CFA_offset: RBP -16
DW_CFA_advance_loc: 3 to 0x19978
DW_CFA_def_cfa_register: RBP
DW_CFA_advance_loc: 6 to 0x1997e
DW_CFA_offset: RBX -40
DW_CFA_offset: R14 -32
DW_CFA_offset: R15 -24
DW_CFA_advance_loc1: 100 to 0x199e2
DW_CFA_def_cfa: RSP +8
DW_CFA_advance_loc: 5 to 0x199e7
DW_CFA_def_cfa: RBP +16
```

DWARF Unwind Info: Problems

1. Significant **size overhead**
   - libLLVM.so (135 848 kiB): 6 203 kiB `.eh_frame` + 950 kiB `.eh_frame_hdr`
     $\rightsquigarrow$ 5.2% overhead

2. **Bytecode interpretation is slow** and takes unpredictably long
   - Costly for every exception
   - Prevents practical use for stack tracing in profilers

Compact Unwind Information: State of the Art

- Apple Compact Unwind: one 32-bit descriptor per function
  only synchronous unwind info $\rightsquigarrow$ no full replacement
- OpenVMX's extension: imprecise at prologue/epilogue

- Microsoft pdata/xdata: async; compact info limited to simple funcs
  imposes rigid constraints on instrs in prologue/epilog; AArch64 only

- SFrame: stack tracing only, no callee-saved regs/exception info
  also not really compact...

Async Compact Unwind Descriptors I

Goal: accurately describe >95% of existing code generated by Clang/GCC

- 64-bit descriptor encodes info on prologue and epilogue size/content
  - Support typical stack frame sizes/layouts and callee-saved reg orders
- Permit multiple descriptors per function
  - Required for tail duplication: generally one descriptor per tail
- Descriptors can encode any state inside prologue/epilogue
  - Required for GCC's instruction schedules

Async Compact Unwind Descriptors I

Goal: accurately describe >95% of existing code generated by Clang/GCC

- 64-bit descriptor encodes info on prologue and epilogue size/content
  - Support typical stack frame sizes/layouts and callee-saved reg orders
- Permit multiple descriptors per function
  - Required for tail duplication: generally one descriptor per tail
- Descriptors can encode any state inside prologue/epilogue
  - Required for GCC's instruction schedules

$\Rightarrow$ Can encode >99.93% of libLLVM.so functions at -O3

- Unsupported function can always use DWARF as fallback

Async Compact Unwind Descriptors II

- Object file encoding: replace bytecode with unwind descriptors
  - Indicated by new `C` augmentation in FDE
  - Provides good compatibility with existing linkers
  - Implemented in LLVM-MC (integrated assembler) for x86-64
  - `.eh_frame` size reduction for LLVM objects: 30%

Async Compact Unwind Descriptors II

- Object file encoding: replace bytecode with unwind descriptors
  - Indicated by new `C` augmentation in FDE
  - Provides good compatibility with existing linkers
  - Implemented in LLVM-MC (integrated assembler) for x86-64
  - `.eh_frame` size reduction for LLVM objects: 30%
- Executable/shared lib encoding: modified Apple-style encoding
  - Two-level tables with dictionary encoding in `.eh_frame_hdr` (v2)
  - `.eh_frame` only needed for DWARF fallbacks

Async Compact Unwind Descriptors II

- Object file encoding: replace bytecode with unwind descriptors
  - Indicated by new `C` augmentation in FDE
  - Provides good compatibility with existing linkers
  - Implemented in LLVM-MC (integrated assembler) for x86-64
  - `.eh_frame` size reduction for LLVM objects: 30%
- Executable/shared lib encoding: modified Apple-style encoding
  - Two-level tables with dictionary encoding in `.eh_frame_hdr` (v2)
  - `.eh_frame` only needed for DWARF fallbacks

$\Rightarrow$ **86% size reduction** on `.eh_frame_hdr`+`.eh_frame` for libLLVM.so