Compact Unwind Information for ELF Platforms

Alexis Engelke
alexis.engelke@inria.fr

GNU Tools Cauldron, Prague, CZ, 2026-10-02

Inria

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 (192 MiB, 81 MiB `.text`):
     7 180 kiB `.eh_frame` + 1 138 kiB `.eh_frame_hdr` $\rightsquigarrow$ 4.2% file size 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, codegen constraints
- OpenVMS's extension: imprecise at prologue/epilogue
- Microsoft pdata/xdata: async; AArch64 compact info limited to simple funcs
  imposes rigid constraints on instrs in prologue/epilog
- SFrame: stack tracing only, no callee-saved regs/exception info
  also not really compact...

Async Compact Unwind Format

Goal: accurately describe 99% of existing code generated by Clang/GCC

- ∼lossless compression of existing DWARF unwind info
- Permit several descriptors per function
  - Required for shrink wrapping, tail duplication, GCC's instr scheduling
- Support typical stack frame sizes and CSR reg layouts
- Fallback to DWARF FDE possible

Descriptors for Each Row

- Step 1: one descriptor must describe each single unwind info row

```
    f1:
 0:   push rbx                  rsp, sz=8, csr=∅
 1:   sub rsp, 0x20             rsp, sz=16, csr=rbx
 5:   mov rdi, rsp              rsp, sz=48, csr=rbx
 9:   call foo
 f:   add rsp, 0x20
13:   pop rbx                   rsp, sz=16, csr=rbx
14:   ret                       rsp, sz=8, csr=∅
15:   cs nopw [rax+rax*1+0x0]
```

Descriptors with Prologue/Epilogue Compression

- Step 2: descriptor can imply common prologue/epilogue sequence
  - Descriptor encodes prologue/epilogue size
  - epilogue size relative to start address of next descriptor

```
f1:
 0:    push rbx                    rsp, sz=48, csr=rbx, psz=5, esz=0xd
 1:    sub rsp, 0x20
 5:    mov rdi, rsp                (end of prologue)
 9:    call foo
 f:    add rsp, 0x20
13:    pop rbx                     (begin of epilogue)
14:    ret
15:    cs nopw [rax+rax*1+0x0]
20:                                (end of descriptor range)
```

Multiple Descriptors

```
     f2:
 0:    push r14                  rsp, sz=16, csr=r14, psz=2, esz=0
 2:    mov r14, rdi              (end of prologue)
 5:    push r13                  rsp, sz=32, csr=r14,r13,r12, psz=4, esz=2
 7:    push r12
 9:    ...                       (end of prologue)
80:    pop r12
82:    pop r13                   (begin of epilogue)
84:    xor eax, eax              rsp, sz=16, csr=r14, psz=0, esz=1
86:    pop r14
88:    ret                       (begin of epilogue)
89:    call ...                  rsp, sz=32, csr=r14,r13,r12 psz=0,esz=0x10
8e:    pop r12
90:    pop r13                   (begin of epilogue)
92:    pop r14
94:    ret
95:    cs nopw [rax+rax*1+0x0]
```

Descriptor Encoding (x86-64)

```
struct DescRSP : uint64_t {
  rbp,r15,r14,r13,r12,rbp,rbx : 1
  // CFA=RSP-x, max: 4 MiB
  frame_size : 19
  personality: 3
  mode : 3 = 1 // RSP desc
  psz : 8
  esz : 8
  GNU_arg_size : 8
  reserved : 8
}
```

```
struct DescRBP : uint64_t {
  rbp,r15,r14,r13,r12,rbp,rbx : 1
  reserved : 11
  inner_psz : 8
  personality: 3
  mode : 3 = 2 // RBP desc
  psz : 8
  esz : 8
  GNU_arg_size : 8
  reserved : 8
}
```

- RBP can be saved at different positions (GCC vs. LLVM) $\Rightarrow$ two bits
- Modes for "no frame" (0) and FDE escape (7)
- Enough space for future extensions (APX, clash protection, ...)

.eh_frame_hdr v2

- Apple-inspired two-level encoding
  - Avoids binary search over large area
- Descriptors deduplicated at global and page level
- Top-level structure:
  - Version, encodings, personality pointers, global descriptors
  - Page search table: start address, page offset in section, LSDA table offset
- Page: max 4 kiB (somewhat arbitrary), covers max. 1 MiB of code
  - Sorted 32-bit entries: start offset (20 bit), descriptor idx (12 bit)
  - List of page-local descriptors

Unwind Info Size `.eh_frame_hdr+.eh_frame`

x86-64, libLLVM.so, Clang 23, GCC 15, -O3

	DWARF EH	CU (no p/e)	Compact Unwind	EH-Sz	File-Sz
GCC	10 647 kiB	6 057 kiB	3 099 kiB	-70.9%	-3.34%
	(1488+9158)	(6049+7)	(3092+7)		
w/exceptions	10 859 kiB	6 185 kiB	3 367 kiB	-69.0%	-3.00%
	(1589+9270)	(6177+8)	(3359+8)		
Clang	8 318 kiB	5 036 kiB	1 080 kiB	-87.0%	-3.67%
	(1138+7180)	(5036+0)	(1080+0)		
w/exceptions	9 403 kiB	6 284 kiB	2 042 kiB	-78.3%	-3.49%
	(1218+8184)	(6284+0)	(2042+0)		

- GCC fallbacks due to stack clash protection (fixable) and DRAP
- GCC primarily larger due to instruction scheduling
- SFrame v2: 11 410 kiB

Other Architectures: AArch64

- Design work in progress
- Difficulty: SVE
  - Dynamic CFA/CSR offsets uses DWARF expressions, needs analysis
  - Many more CSRs: 11 GP, 16 vector, 12 predicate regs
    already 1 bit per CSR requires practically entire 64-bit encoding space
- Minor difficulty: GCC's stack frame layouts
  - CSRs not stored consecutively, not always in order
  - All representable, but costs encoding space

Implementation

- Implementation in linker, unwinder, (readelf) only
  - No changes to assembler, compiler, object file format
  - Changes to compiler might yield better results
- Linker-side is a bit complex and architecture-dependent
  - Unavoidable, architecture-knowledge required for good compression
  - Previous implementation in assembler proved more complex and requires a separate format in the object file

- Implementation in LLD: 1 kLOC (0.3 kLOC x86-64 specific)
- Implementation in llvm-readelf: 0.3 kLOC

Future Work / Open Questions

- Future work: libgcc/libunwind implementation, evaluate C++ unwind perf
- Future work: in-kernel backtrace support for perf samples
- (I don't expect glibc changes are necessary?)

- Design feedback wanted!
- Linker complexity? (also cf. sframe discussions)
  - Linker needs to understand `.eh_frame` already
- GCC codegen changes for more compact unwind info?
- How to avoid format fragmentation?
  - Meta started shipping custom compact format, ...

Compact Unwind: Summary

- Unwind info: meta-information needed for stack unwinding/tracing
- Bytecode-based DWARF format is large and slow to interpret

- Proposal: adopt new async compact unwind info format for GNU/ELF
- ∼lossless encoding with 69–87% size reduction on x86-64
  while maintaining efficient access
- Fallback to DWARF for unsupported encodings possible

Thanks to Fangrui Song and Shoaib Meenai for their feedback.