What’s possible in observability when we have frame pointers

Matthias Loibl / Jon Seager
Continuous Profiling

What happened at the spikes?!
Continuous Profiling
About us

Matthias Loibl
- Senior Software Engineer at Polar Signals
- Open-Source Maintainer
  - Parca
  - Thanos
  - Prometheus
  - Prometheus Operator
  - Pyrra

@metalmatze@social.metalmatze.de
@metalmatze

Jon Seager
- VP Engineering at Canonical
- Leads development of Juju & charms
  - Observability
  - Identity
  - Data Platform
  - MLOps
  - Telco

@jnsgruk@hachyderm.io
@jnsgruk
What is profiling data made up of?

```
t1
 a
 b
 c
 d

 t2
 a
 b
 c
 e

 t3
 a
 b
 c
 d
```

\[=\]

```
a;b;c;d 20ms
a;b;c;e 10ms```

How do we get a stack?

a
b
c
d
Best Case: Frame Pointers
What are frame pointers?

```c
int top(void) {
    for(;;) { }
}
int c1(void) {
    top();
}
int b1(void) {
    c1();
}
int a1(void) {
    b1();
}
int main(void) {
    a1();
}
```

# compiled with `gcc sample.c -o sample_with_frame_pointers -fno-omit-frame-pointer`
$ objdump -d ./sample_with_frame_pointers

```asm
0000000000401106 <top>:
    55                      push   %rbp
    48 89 e5                mov    %rsp,%rbp
    eb fe                   jmp    40110a <top+0x4>

000000000040110c <c1>:
    55                      push   %rbp
    48 89 e5                mov    %rsp,%rbp
    e8 f1 ff ff ff          call   401106 <top>
    90                      nop
    5d                      pop    %rbp
    c3                      ret

0000000000401118 <b1>:
    55                      push   %rbp
    48 89 e5                mov    %rsp,%rbp
    e8 eb ff ff ff          call   40110c <c1>
    90                      nop
    5d                      pop    %rbp
    c3                      ret

...
Binary

Executable Code bytes/instructions in Binary

ra - 1 is our caller
Walking the Stacks

- Core registers
  - Keep track of process state
    - PC (program Counter)/ IP (Instruction Pointer)
    - rbp/fp: base pointer/frame pointer
    - rsp/sp: stack pointer
    - ra: return address
Using frame pointers in eBPF
What’s eBPF?

No syscalls

sendmsg()  recvmsg()

No syscalls

Sockets

TCP/IP

Network Device
Get a stack in BPF

- **bpf_get_stack**
  - BPF helper to unwind the stack using frame pointers

```c
bpf_user_pt_regs_t *regs = &ctx->regs;
u64 ip = PT_REGS_IP(regs); // read leaf instruction pointer
u64 bp = PT_REGS_FP(regs); // read leaf base pointer
u64 ra = 0; // return address

// *save leaf frame*
for (int i = 0; i < MAX_STACK_DEPTH; i++) {
  // return address is the next register from rbp, so 8 bytes away
  err = bpf_probe_read_user(&ra, 8, (void *)bp + 8);
  if (err < 0) {
    // error
  }
  // Rewinding the program counter to get the instruction pointer for the
  // previous function would be ideal but is unreliable in `x86` due to
  // variable width encoding. We can ensure correctness only by disassembling
  // the `.text` section which would be unfeasible. Since return addresses
  // always point to the next instruction to be executed after returning from
  // the function (and stack grows downwards), subtracting 1 from the current
  // `ra` gives us the current instruction pointer location, if not the exact
  // instruction boundary
  ip = ra - 1;

  // *save frame*
  // read content of base pointer into bp variable
  err = bpf_probe_read_user(&bp, 8, (void *)bp);
  if (err < 0) {
    // error
  }
  // if bp == 0 we've reached the bottom of the stack
}```
Having frame pointers in BPF makes regular profiling easy
Why do frame pointers matter for observability?

- **Simplified Profiling:**
  - Don't worrying about compiler configurations
- **Lower Overhead:**
  - Cheaper than using DWARF or DWARF-derived information to unwind
- **Debugging Accessibility:**
  - bcc-tools, bpftrace, perf and other such tooling to work out of the box

Check out last year's FOSDEM talk! "Stack walking/unwinding without frame pointers"
Possibilities are endless:
Unwinding takes 2 memory reads

- bpftrace
  - `ustack builtin`, the `bpf_get_stack` in bpftrace’s language
  - `bpftrace -e 'profile:hz:99 { @[ustack] = count(); }'` // one-liner profiler
- Go execution tracer
- Profile Guided Optimizations (PGO) research
  - Context-sensitive sampling-based PGO (CSSPGO)
Communicating with Userspace

- Hook
- eBPF program
- eBPF Maps
- Parca Agent

Uses frame pointers

10s
Bringing frame pointers to the masses.
Ubuntu 24.04 LTS will have frame pointers enabled by default on 64-bit platforms.
Performance implications and future plans for optimisation.
Frame pointers are just the start.
Canonical is building a company-wide Performance Engineering machine.
snap install parca

snap install parca-agent --classic

juju deploy parca[-k8s]
Get in touch!

Matthias Loibl
@metalmatze@social.metalmatze.de
@metalmatze

Jon Seager
@jnsgruk@hachyderm.io
@jnsgruk