gcc -finstrument-functions

```
        .section        .rodata.str1.1,"aMS",@progbits,1
.LC0:
        .string "Hello world!"
        .section        .text.startup,"ax",@progbits
        .globl  main
        .type   main, @function
main:
        subq    $8, %rsp
        movl    $main, %edi
        movq    8(%rsp), %rsi
        call    __cyg_profile_func_enter
        movl    $.LC0, %edi
        call    puts
        movq    8(%rsp), %rsi
        movl    $main, %edi
        call    __cyg_profile_func_exit
        xorl    %eax, %eax
        popq    %rdx
        ret
```

gcc -pg

```
        .section        .rodata.str1.1,"aMS",@progbits,1
.LC0:
        .string "Hello world!"
        .section        .text.startup,"ax",@progbits
        .globl  main
        .type   main, @function
main:
        pushq   %rbp
        movq    %rsp, %rbp
1:      call    mcount
        movl    $.LC0, %edi
        call    puts
        xorl    %eax, %eax
        popq    %rbp
        ret
```

Issues with Instrumentation-Based Profiling

- Requires compiler support
- Changes performance characteristics:
  inflates cost of tiny, frequently called functions

```
// Linux kernel include/linux/compiler.h
#ifdef CONFIG_PROFILE_ALL_BRANCHES
/*
 * "Define 'is'", Bill Clinton
 * "Define 'if'", Steven Rostedt
 */
#define if(cond, ...) __trace_if( (cond , ## __VA_ARGS__) )
#define __trace_if(cond) \
        if (__builtin_constant_p(!!(cond)) ? !!(cond) :                 \
        ({                                                              \
                int ______r;                                            \
                static struct ftrace_branch_data                        \
                        __attribute__((__aligned__(4)))                 \
                        __attribute__((section("_ftrace_branch")))      \
                        ______f = {                                     \
                                .func = __func__,                       \
                                .file = __FILE__,                       \
                                .line = __LINE__,                       \
                        };                                              \
                ______r = !!(cond);                                     \
                ______f.miss_hit[______r]++;                            \
                ______r;                                                \
        }))
```

Instrumentation with Coccinelle

Coccinelle (“semantic patch”) automates source transforms:

instr.cocci:

```
@@
expression init, test, step;
statement stmt;
@@
- for
+ { _trace_start(); for
  ( init;
- test;
+ _trace_test(test);
  step) stmt
+ }
```

```
spatch -sp_file instr.cocci in.c -o out.c
```

Sampling Profiling

CPUs have PMU events:

- Clock cycle
- Instruction issue
- Branch, mispredicted branch
- Cache hit/miss (L1/L2/LLC)
- Execution ports utilization
- (etc, many CPU-specific events)

(caveat: only up to 4-8 can be active together)

OS can setup PMU to:

- Count events (“samples”)
- Trigger interrupts after N samples

Linux perf tool

Basic commands:

- Interactive system-wide view: perf top
- Overview for one commad: perf stat
- Profiling run: perf record
- Profiling view: perf report
- List events: perf list
  note: includes software events!

perf gotchas

- Sampling skid (add `:pp` suffix to events)
- Cache eviction effects:
  after one function wipes most of cache,
  other code pays the cost of misses

perf helpers

Andi Kleen's pmu-tools:

- ocperf.py: more PMU events
- toplev.py: guided top-down investigation
  ISPASS 2014 paper: *A Top-Down Method for Performance Analysis and Counters Architecture, Ahmad Yasin*, URL

Brendan Gregg's linux-perf page:

- Tips, one-liners
- Flamegraphs!