Finding Missed Compiler Optimizations

Gergö Barany - Inria Paris, France

... By differential testing

even tiny examples show optimization differences:

Source

```
int f(int p, int q) {
  return q + (p % 6) / 9;
}
```

Clang

```
movw r2, #43691
movt r2, #10922
smmul r2, r0, r2
add r2, r2, r2, lsr #31
add r2, r2, r2, lsl #1
sub r0, r0, r2, lsl #1
movw r2, #36409
movt r2, #14563
smmul r0, r0, r2
asr r2, r0, #1
add r0, r2, r0, lsr #31
add r0, r0, r1
bx lr
```

GCC

```
mov r0, r1
bx lr
```

Clang did not know that `(p % 6) / 9 = 0` on ints, fixed now

Method

* generate random C programs
* compile with different compilers
* compare: count instructions of interest
* reduce to minimal example

... or more directly

Source

```
int g(short p, double q) {
  int a = 0;
  if (p)
    a = (int) q;
  return a;
}
```

GCC

```
cmp r0, #0
sub sp, sp, #8  ⇐ useless stack frame
vcvtne.s32.f64 s15, d0
streq r0, [sp, #4]  ⇐ useless spill
vmovne r0, s15
vstrne.32 s15, [sp, #4]  ⇐ useless spill
add sp, sp, #8  ⇐ useless pop
bx lr
```

dead stores never reloaded, don't correspond to anything in the source

found by liveness analysis on the binary

some results

tested GCC, Clang, and CompCert

found missed arithmetic optimizations in each

and cases of: Bad spilling, Bad coalescing, dead stores, redundant computations, missing instruction selection patterns

some reported, some fixed

more examples, preprint

https://github.com/gergo-/missed-optimizations