Google

Asm Goto with Outputs

2020 LLVM Virtual Developers' Meeting

Bill Wendling & Nick Desaulniers

Motivation

Asm goto use cases

- Exceptions—fault handler fixups
- Tracing—replacing branches (original use case)
- Runtime devirtualization

Curiously recurring inline asm .pushsection

This pattern occurs throughout the Linux kernel:

```
        asm goto(".pushsection foo"
                 ".long %l0"
                 ".popsection"
                 : : : : comefrom);
        /* ... */
comefrom:;
```

By storing the address of a label into a different ELF section (via inline asm), if we have the machinery to parse ELF sections, then we may revisit the statement (or otherwise store information) about our program to find at runtime.

Interrupts & Exceptions

“Vectored events” where the CPU may be able to backup register state to memory for recovery then jump to registered handler routines.

- Interrupts
  - Maskable (ignorable)
  - Non-maskable
- Exceptions
  - Aborts (unable to proceed)
  - Traps (debugging, kernel fp handling; increments program counter)
    - Software interrupts or “programmed exceptions”
  - Faults (potentially recoverable)

Exceptions (fault handler fixups)

Example from `arch/x86/include/asm/uaccess.h` for writing to syscall arguments from userspace, (simplified, see also `Documentation/x86/exception-tables.rst`):

```
#define __put_user_goto(x, addr, label)         \
    asm volatile goto(                          \
      "1: mov %0,%1"                            \
      ".pushsection __ex_table"                 \
      ".long 1b"                                \
      ".long %l2"                               \
      ".long ex_handler_uaccess"                \
      ".popsection"                             \
      : : "ir"(x), "m"(addr))                   \
      : : label)
```

where `addr` comes from userspace (i.e. can't trust), might fault if not paged in or is invalid.

Exceptions (fault handler fixups)

Example from `arch/x86/kvm/vmx/vmx.c` (simplified):

```
int kvm_cpu_vmxon(long vmxon_pointer) {
      asm volatile goto("1: vmxon %[vmxon_pointer]\n\t"
                        ".pushsection __ex_table\n"
                        ".long 1b\n"
                        ".long %l[fault]\n"
                        ".long ex_handler_uaccess\n"
                        ".popsection\n"
                        : : [vmxon_pointer] "m"(vmxon_pointer)
                        : : fault);
      return 0;
fault:
      printk("oh no!\n");
      return -EFAULT;
}
```

Tracing

Motivating example: We want on rare occasions to call the `trace` function; on other occasions we'd like to keep the overhead to the absolute minimum. We can patch the *nop* instruction (“nop sled”) at run time by finding data stored in this section to be an *unconditional branch* to the stored label.

```
#define TRACE1(NUM)                                 \
    do {                                            \
         asm goto ("0: nop;"                        \
                   ".pushsection trace_table;"      \
                   ".long 0b, %l0;"                 \
                   ".popsection"                    \
                   : : : : trace#NUM);              \
         if (0) { trace#NUM: trace(); }             \
    } while (0)

#define TRACE TRACE1(__COUNTER__)
```


BUT WAIT
THERE'S MORE

Devirtualization

If we could runtime patch conditional jump instructions in or out, what else could we replace at runtime?
How about turning indirect calls that change infrequently or not at all into direct calls? (Relief from Spectre)
Pretty dangerous; requires at least icache flushes, trickier for variable length encoded ISAs (hint: nop sleds!).

Nitty-Gritty Details

Overarching goals

- Allows `asm goto` to behave the same as a normal asm block on the default / fallthrough path.
- Allows the programmer to optimize code further:
  - No longer need to use memory for outputs.
  - Improve the programmers ability to reuse labels as exceptional cases.
  - Reduce the amount of generated code—e.g. `unsafe_get_user()`.

Ambiguous cases

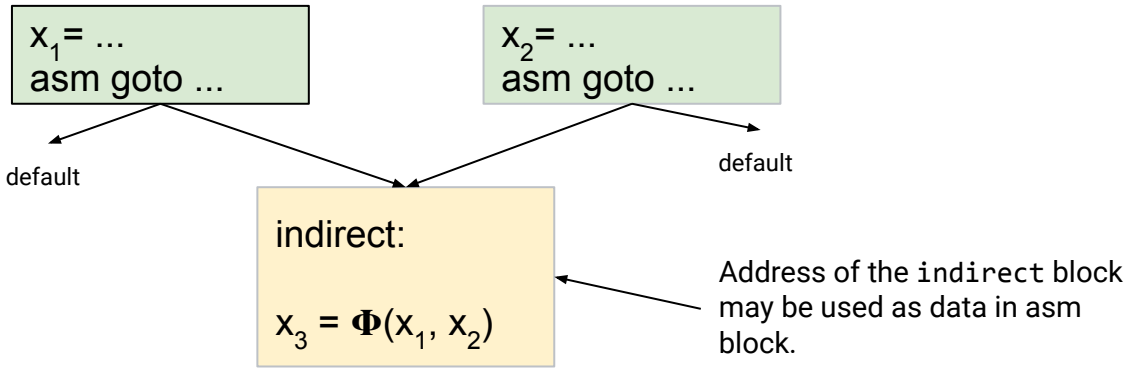
1. Multiple `asm goto` statements with the same target, but non-mutually satisfiable output constraints.
   a. I maintain that `asm goto` statements *shouldn't* jump to the same basic block, but normal transformations may make it impossible to enforce that assertion.
2. Jumping to labels where the output variable is out of scope.
   a. Shouldn't be able to refer to out of scope variables, but maybe something gross like this.

```
int foo() {
    int y;
    asm goto("..." : "=r"(y) : : : label);

    int x = bar();
    if (0) { label: y = x; }
    return y;
}
```

Asm goto with outputs details

- GCC didn't implement asm goto with outputs, due to an internal restriction.
- In our implementation, outputs are supported only on the fallthrough path.
  - Supporting outputs on the indirect branches is very messy. E.g. it's not clear how to resolve PHI nodes when a destination block has its address taken.



Design details

- `callbr` is converted to `INLINEASM_BR` in MIR (**M**achine **IR**)
  - MIR allows for multiple terminators at the end of blocks.

Design details

- `callbr` is converted to `INLINEASM_BR` in MIR (**M**achine **IR**)
  - MIR allows for multiple terminators at the end of blocks.
- ASM goto's representation as a terminator in MIR didn't fit well with clang's back-end restrictions—i.e. there cannot be a non-terminator after a terminator.

Design details

- `callbr` is converted to `INLINEASM_BR` in MIR (**M**achine **IR**)
  - MIR allows for multiple terminators at the end of blocks.
- ASM goto's representation as a terminator in MIR didn't fit well with clang's back-end restrictions—i.e. there cannot be a non-terminator after a terminator.
  - Difficult to represent moving values from an asm goto call into registers before the end of the block, because there cannot be non-terminators (`MOV` instructions) after terminators.
    - Could place moves in separate fallthrough block, but "live in" analysis isn't ran until late in MIR processing.

Design details

- `callbr` is converted to `INLINEASM_BR` in MIR (**M**achine **IR**)
  - MIR allows for multiple terminators at the end of blocks.
- ASM goto's representation as a terminator in MIR didn't fit well with clang's back-end restrictions—i.e. there cannot be a non-terminator after a terminator.
  - Difficult to represent moving values from an asm goto call into registers before the end of the block, because there cannot be non-terminators (`MOV` instructions) after terminators.
    - Could place moves in separate fallthrough block, but "live in" analysis isn't ran until late in MIR processing.
  - Live range splits may need to spill after an asm goto, resulting again in a non-terminator after terminator violation.

Design details

- `callbr` is converted to `INLINEASM_BR` in MIR (**M**achine **IR**)
  - MIR allows for multiple terminators at the end of blocks.
- ASM goto's representation as a terminator in MIR didn't fit well with clang's back-end restrictions—i.e. there cannot be a non-terminator after a terminator.
  - Difficult to represent moving values from an asm goto call into registers before the end of the block, because there cannot be non-terminators (`MOV` instructions) after terminators.
    - Could place moves in separate fallthrough block, but "live in" analysis isn't ran until late in MIR processing.
  - Live range splits may need to spill after an asm goto, resulting again in a non-terminator after terminator violation.
- Ultimately, we decided that the asm goto representation in MIR *shouldn't* be a terminator (thanks, James!).
  - However, we must ensure that uses of non-output variables on the indirect branches are defined *before* the asm block.

Finally, the end!

- Clang-built Linux (https://clangbuiltlinux.github.io/) is a renewed effort to make clang a first-class citizen in the Linux world.
- It's mutually beneficial for the gcc and clang communities to collaborate on Linux support.
- Both compilers bring different things to the table:
  - Warnings, sanitizers, code health tools, ideas for language extensions, etc.

One more thing...

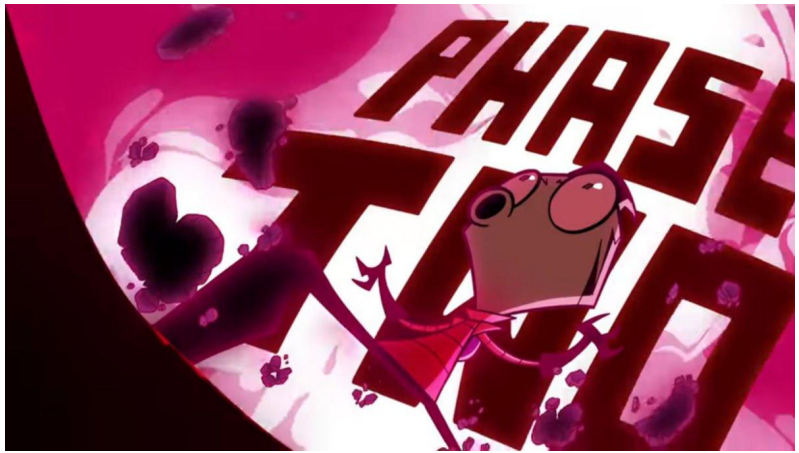
AGwO and beyond the infinite

Linux: Commit [587f17018a2c]

```
Kconfig: add config option for asm goto w/ outputs
```

tcmalloc: Commit [https://github.com/google/tcmalloc/commit/ca9fa6e5a5b283eebcf008ba081491a0d946f57d]

```
Leverage asm goto with output to optimize new fast path further.
```



Also...

We're running Clang-built Linux at Google now!