Tuning in-kernel routines

VRULL GmbH
Heiko Stübner

September, 2022
Whoami

- Kernel developer at VRULL
- Recent RISC-V work SVPBMT, Zicbom DMA operations
- So spent way too much time on code patching
What are we talking about

- Routines normally implemented by C-libraries
- Strings: strcmp, strncmp, strlen
- Memory: memcpy, memcmp, memset, memmove
- Kernel of course needs its own implementation
- lib/string.c provides very generic variants
Architecture specific

- Generic code protected by #ifndef __HAVE_ARCH_*
- Architecture can override with optimized variants
- Assumes either generic or architecture-specific
The RISC-V “problem”

- Small set of core instructions + extensions (existing, planned, future)
- Mix and Match
- Cores can / cannot support any number of them
- Results in possibly many variant-implementations
- With different performance characteristics
Examples

- Things I’ve seen wip-implementations for
- `strcmp`: zbb+unaligned > zbb > generic (main kernel)
- `strlen`: zbb > generic (main kernel)
- `strncmp`: zbb > generic (main kernel)
- `memcpy`: rv64+unaligned > generic (riscv)
- `memmove`: rv64+unaligned > generic (riscv)
- `memset`: rv64+unaligned+cboz > rv64+unaligned > generic (riscv)
Fast unaligned access

- Some cores can do unaligned access fast
- Kernel currently assumes unaligned access is always slow
- Not really an ISA feature
- So not detectable with current means

```
KBUILD_CFLAGS += $(call cc-option,-mstrict-align)
```

- Reason given:
  they're emulated by machine mode traps on all extant architectures. It's faster to have GCC emit only aligned accesses.
Glibc has ifunc

- libc_ifunc (__libc_strcmp,
  HAVE_RV(zbb) && HAVE_FAST_UNALIGNED()
  ? __strcmp_zbb_unaligned
  : HAVE_RV(zbb)
    ? __strcmp_zbb
    : __strcmp_generic);

- Resolver runs at startup, selects variant the app will use
Just some static keys?

- If `(static_branch_unlikely(zbb_and_fast_unaligned))`
  
  ```c
  Return __strcmp_zbb_unaligned();
  ```

- If `(static_branch_unlikely(zbb))`
  
  ```c
  Return __strcmp_zbb();
  ```

- Return `__strcmp_generic()`;

- Caveat: Static keys allows the inclusion of seldom used features [...] 

- Penalizes the “unlikely” variant
Alternatives

- `asm(ALTERNATIVE("jal __strcmp_generic", ...
  "Jal __strcmp_zbb", ...));`
- Penalty-free swap to other variant
- Caveat: probably undefined behaviour when similar features supported 
  (i.e. zbb but also zbb+fast_unaligned)
Questions to solve

- Detecting non-ISA features (fast unaligned access)
- -> Unified discovery ?
- Re-use generic functions in some way?
- How to do selection (table with implementations sorted by performance)