Jan 18, 2024 | RISC-V Perf Analysis SIG Meeting

Attendees: tech.meetings@riscv.org Beeman Strong Marc Casas

Notes

- **Attendees**: Snehasish, Beeman, Greg, Bruce, Robert, Suraj, MattT, Dmitriy, Fei, Min, Ved, Chun
- **Slides/video** here
- Technical issues with zoom, moved to Meet. Started late.
- **Talk: PMU based profiling in the datacenter by Snehasish Kumar**
- GWP (Google-Wide Profiler) samples <5% of machines at a time, keep overhead <5%
- Collecting call-stack with samples is the key part of a key/value DB
  - Done in interrupt handler
- Also collect uncore BW, though not tied to an application
- 100s of MB of binaries, can't always include debug info, so have step to associate raw profiles with symbols
- Stack unwinding critical, even though it costs 1-2% perf (for frame pointers)
  - This is from x86. Will it be similar for RISC-V? May be less, since RISC-V isn't as register-starved.
  - Would make profiler life much easier if we can resolve these issues with stack collection
- Stack unwind alternatives:
  - Dwarf unwinder too costly for Google to keep info
  - Kernel unwinder similar, stores some info in the binary. SFrame similar as well, adds some metadata overhead. Would need this info for every function in the binary, for large-footprint apps it's impractical.
- LBR call-stack depth of 32 is not enough, pre-existing stack is missing
  - Could pre-load LBR to deal with pre-existing stack
  - LBR not available throughout the fleet
- On spec, CET overhead for shadow stack was ~0.8%. RISC-V should be even less, don't have to push/pop for leaf functions.
  - Google's measurements for CET were in low single digits
  - Haven't really used this yet.
- Could a hash of the callstack be used as a key?
  - Have used that as an optimization, but would have to be reversible
- What about language's that switch stacks, like Silk and Go?
  - Go has own profiling, don't use counters there
  - Everything discussed here is for C++
- LBR/BRS is critical part of autofdo, get ~11% over -O2
  - With instrumentation-based FDO get 14-15%. But can be 2x overhead, can't use it in production. And hard to simulate production workload synthetically.
  - Even 16-deep LBR gets most of the benefit. 32 is sweet spot, diminishing returns above that.

- Count near_taken branches, which does match what is recorded in LBRs but that's not critical. Just want to avoid biases, non-determinism.
- Google uses instrumentation PGO for one thing, autofdo for everything else
    - Will publish modified autofdo at CGO this year
- ASMDB: use control flow info to build database of dynamic inst execution
    - Used for fine-grained function splitting, to separate hot and cold components
    - Includes cycle count info from LBR
- Dispatch sampling (aka instruction sampling)
    - Biggest problem is sample throughput for infrequent events (e.g., LLC miss)
        - These events contribute to tail latency, very important
    - Hard to guarantee buffer size of 1, so can collect call-stack
        - So is this any better than interrupt sampling?
            - Yes, because get extra metadata, and may get precise interrupt
    - Uop sampling introduces undesirable bias towards microcoded instructions
    - Can't do time-based sampling with this, but Google doesn't do that
- PEBS
    - Here also need interrupt on each record, to get call-stack
        - Could put SS in record, but would require a memcpy
    - Think it would be better to write PEBS buffer to memory, not cache, for cases where not taking an interrupt on each record (large PEBS buffer)
        - Would want streaming stores, so not serialized
- Google has announced Android support on RISC-V, autofdo is used on Android, CTR will be critical

Action items

- ☐ Beeman Strong - Jul 28, 2022 - Reach out about proprietary performance analysis tools
- ☐ Beeman Strong - Jul 28, 2022 - Reach out to VMware about PMU enabling