Aug 22, 2024 | RV Perf Analysis SIG

Attendees: Beeman Strong tech.meetings@riscv.org Snehasish Kumar Chynoweth, Michael W

Notes

- **Attendees**: Beeman, Michael Chynoweth, BjornT, BruceA, DeepakL, Dmitriy, Keeran, LuisP, MattT, Paul Clarke, RobbinE, RobertC, RyanM, SiavashK, Ved, VictorL, GregF, Snehasish, AashishP, VikasV, StephenP
- **Slides/video** here
- Updates
    - CTR nearing end of public review (actually just ended)
    - Self-hosted trace TG about to start
- **Industry talk: Zen and the art of Performance Monitoring and Optimization - Mike Chynoweth**
    - Using TMA to inform hybrid P-core vs E-core decisions
        - Prefer LoadStore_Bound vs Memory_Bound, to avoid confusion with DRAM accesses
    - Example using Blender stack (Classroom), E-core was under-performing
        - Dug into TMA, found large increase in indirect mispredicts on E-core
        - Is PGO used on this software? No, that would have fixed the problem
        - Indirects means jumps with indirect targets (ala JALR), ignoring RETs
        - E-core (Gracemont) has smaller indirect target array
        - BE_BOUND difference resulted in some fixes as well, but several smaller ones
    - Starting to use PMU for Hybrid decision-making
        - Intel Thread Director (ITD) looks at ~16 running events
            - e.g. VNN insts, which have much higher throughput (4x) on P-cores
            - Have dedicated counters for ITD, not sharing programmable counters with other SW
        - There is overhead to migrate across core types, data ends up in L3
        - Video game example before hybrid went to production, hybrid was seeing more L3_BOUND vs non-hybrid
            - Even when found workloads that would run better on P-core, wasn't worth the migration cost
            - Migrating too aggressively, turned it down. Then hybrid outperformed non-hybrid
        - How does ITD make a decision to migrate?
            - Looks at dedicated events, puts workload in one of 4 classes. E.g., spinning often. In that case, move it to E-core. OTOH, if high VNNI, move it to P-core.
    - Unreal engine example

        - 15% of priority gaming releases are Unreal, so it gets lots of focus
        - Slow frames went way up, user-visible
            - Found DRAM_BOUND way up
        - Use LBRs to graph basic block latency
        - Saw huge cycle totals on a single UC load
        - Pushed a SW fix that improved slow frames by 50%
        - Mystery: some loads taking 1000s of cycles, with “UNKNOWN” data source
            - Used PEBS load latency to get latency and data source
        - Was going to graphics VRAM, MMIO “resizable bar”
            - Only intended to do stores, to modify data directly in graphics
            - Stores are “free”, post-retire and weak memory ordering
        - Load was just to check the stored values, removed it
    - Timed LBRs
        - Skylake added cycle counts per entry
        - For the given basic block, see spiky histogram of latencies
        - 40% of time spent in last spike (100+ cycles) that was 5% of samples
        - Spike %s loosely map to map to %s of L1/L2/L3 hit and L3 miss percentages
            - But manually attributing cycles to events is tedious
        - Added event counts to LBRs
            - Now can automatically tag events to spikes
            - Events used are those chosen in programmable counters (PMC0..PMC3)
            - Since limited counters, have to multiplex
            - These are 2-bit saturating counters in LBRs
    - HWPGO example eliminates always taken forward jumps in internal workload
        - 18% improvement
        - Added FWD_TAKEN_JCC event in Lion Cove
        - Also use weighted avg distance to contiguous code, using LBRs
            - Could be useful also for tuning knobs in the compiler
    - Now using HWPGO to go after mispredicts
        - Mispredicts costing 58% of slots
        - Simple branch jumping over 2 insts, converted to CMOV
        - Showing separate TMA mispredict % (in BAD_SPEC) and mispredict_resteer % (in FE_BOUND)
        - HWPGO = HW-assisted PGO, using HW counters rather than instrumentation
        - Went to production in ICC in 2023
            - Automatically pushes case-statement hoisting, and also uses APX for optimizing
        - This example was using ICC, not Propeller
    - Goal is to take PGO far beyond frontend optimizations
        - Want to get into data placement, but tougher than frontend

        - Google also interested in this, looking at profile-guided heap optimizations in LLVM
        - Initially, HWPGO full link-time optimizations causes more DTLB misses
            - Mostly from a single lock instruction
            - Sometimes frontend optimizations can make BE_BOUND look worse, just because % goes up, but overall perf is better
                - In this example, perf was worse
            - Lock was alone on own page
        - When optimization data placement, have to be careful not to introduce false-sharing. Now check HITMs to avoid that.
    - Intermittent issues get tagged nicely using Timed LBR + LBR Event Logging
    - Thoughts on attacking heap data vs static data in sampling?
        - Static and global data is about 5% vs local and heap
        - Heap is by far the largest bottleneck, need to optimize memory allocators
        - With heap you need instrumentation in malloc's and free's, can't just use DLA
            - Intel has used PTWRITE, trace-based instrumentation
            - Also need call-stack, have used LBR call-stack
        - Google also collects call-stack, and instruments memory allocator
        - Shiu Liu at College of William and Mary has some good research in this area, also Joe Devietti at UPenn
            - Joe was detecting false-sharing and automatically fixed it, see slide in backup
    - Collecting call-stack
        - Most of what we discussed today didn't use call-stack
        - But issues in shared libraries (e.g., memcpy) require it
            - Saw example of workload doing many 4-byte memcpys, need to know the caller to replace it with a move
            - Ideally want parameters passed to memcpy as well. Get that from PEBS, which includes GPRs.
        - LBR call-stack is cheap, but usually only get 16 entries, problem if that doesn't get you back to the caller
        - For cheap instrumentation, use PTWRITE, along with an precise event for each call
        - May be able to guess at memcpy parameters based on latency
        - Instrumentation-based PGO in LLVM can do some value profiling, but not used yet in HWPGO
        - Including GPRs in PEBS, where you don't need an interrupt, is much cheaper (~500 cycles vs ~10000 cycles)
            - Though adds a lot of data to each record. PTWRITE is nice because add values only to that event. Insert it into memory allocators, probably would do it at call to memcpy/memset.
            - Google published AutoMemcpy, does something similar

        - Ideally compiler doesn't produce 4-byte memcpys, but sometimes size isn't known at compile time

Action items

- [ ] Beeman Strong - Jul 28, 2022 - Reach out about proprietary performance analysis tools
- [x] ~~Beeman Strong - Jul 28, 2022 - Reach out to VMware about PMU enabling~~
    - For x86, VMware exposes counters but not sampling (PMI or PEBS) or Intel PT to guests