Brainstorming Self-hosted Trace

Beeman Strong
3/4/2024

Background

Self-hosted trace describes the use of trace by in-target software

- E.g., gdb or Linux perf

Non-ISA RISC-V trace extensions are designed with external debug in mind

- There are ways for in-target SW to use them, but it is cumbersome

This talk will propose some parameters for future ISA extension(s) that would simplify self-hosted trace

Simple Self-hosted Application Trace Example

1. Application debugger makes SBI request to enable trace
2. Kernel configures TE to trace user mode
3. Kernel routes trace to buffer in user (virtual) memory
4. TE traces application execution to local memory buffer
5. Kernel saves trace config registers when application swapped out, restores trace config registers when application swapped back in
6. Kernel swaps in trace buffer pages if/when they fault
7. Application completes, debugger decodes and processes trace data

Self-hosted Trace Buffer

Resides in memory

- Other sinks/funnels inaccessible

Considerations:

- Virtually vs physically addressed?
- Access restrictions
- Cacheable or not?
- Circular?
    - And detecting buffer full

Self-hosted Trace Buffer: Addressing

Virtual Memory

1. Avoids special case for trace in memory manager
2. Avoids need to find large, contiguous physical memory allocation

Access Control

- Leverage virtual memory access protections
- Gets complicated when more-privileged SW is tracing less-privileged SW

Physical Memory

1. No page faults
2. Simplifies tracing across contexts

Access Control

- Buffer memory is not mapped into page tables for less-privileged SW

Virtual Address Access Considerations

Do trace writes obey virtual memory page protections?

- If so, buffer pages must be RW by the mode being traced
- Can’t be used by, e.g., hypervisor tracing guest(s)

For cases where buffer shouldn’t be accessible by the traced mode

1. Could require use of physically addressed buffer
2. Could use different page tables for trace vs for other memory accesses

Must ensure that trace in internal TE buffers is preserved on page fault

- Implies that page fault handler is not traced, otherwise likely to overflow

Self-hosted Trace Buffer: Cacheability

If trace writes are cacheable:

1. Trace may displace cache lines used by the workload traced, resulting in greater performance overhead
2. Trace collection will be faster

For many uses, trace buffer may wrap multiple times before collection

- Thus the benefit of #2 may be much less than the cost of #1

I suspect we'll want non-cached, non-serialized writes for most cases, but could be configurable

Self-hosted Trace Buffer: Wrap

For many uses, software will want some indication of buffer wrap

- E.g., so that buffer contents can be written out to disk
- Typically this is done via interrupt

To avoid trace loss, interrupt would be signaled before wrap occurs

- Though signaling when the buffer is > half full get complicated

May also want option to stop trace once buffer is full (disallow wrap)?

Self-hosted Trace Config Interface

Probably want a CSR interface to TE and Buffer config

- MMIO could work, but very likely slower
- Any access from outside the hart needs to be disabled by default
  - Debug mode could enable full MMIO ifc
- Likely only need to expose a small set of TCI regs/fields to SW

Need a “TE flush” instruction

- Flushes any trace buffered within TE out to sink
  - Perhaps polling on teTrEmpty is acceptable here
- SW can use this to ensure that trace can’t leak across contexts or trust boundaries

Self-hosted Trace Config: Priv Modes

Privilege Mode filtering required

- So that S-mode user can’t enable trace in M-mode, VS-mode user can’t enable trace in HS-mode, etc

Need hooks to hide the presence of virtualization from a guest using trace

- E.g., vstrcctl.S bit enables trace in VS-mode, vstrcctl.U enables VS-mode
- Any priv mode or context info in the trace appears as though running on bare metal

Self-hosted Trace Config: Other

Need a discovery mechanism for trace capabilities

- WARL probably works for most
- Timestamp source should probably be required to be mtime
- May hardcode some options for self-hosted use (e.g, trTeSrcId)

Additional trace sources:

- Likely wait on data trace
- Consider adding SW trace instrumentation, perf counter trace
- Perhaps use same buffer extension for future perf sampling extension