Arm Architecture 2020 Extensions

Martin Weidmann
Director Product Management, ATG ARM
LVC20-214
Annual cadence: evolving the CPU architecture

Crypto

FP

AArch32

AArch64

SIMD

VHE

Atomics

52-bit addresses

Nested Virt.

52-bit addresses

Nested Virt.

RAS

Stat Profiling

RCpc

Secure EL2

Crypto ++

Predictor Invalidates

BF16 & GEMM

Branch Target IDs

1GHz Timer

Mem Model

Memory Tagging

Fine grain traps

Armv8.0-A

Armv8.1-A

Armv8.2-A

Armv8.3-A

Armv8.4-A

Armv8.5-A

Armv8.6-A

2014

2015

2016

2017

2018

2019
What’s new in 2020

Armv8.7-A

• Support for 52-bit addresses with 4K and 16K granules
• Enhanced support for PCIe hot unplug
• Atomic store operations for interacting with accelerators
• WFE/WFI with timeouts
• Improvements to PAN

Future Architecture Technologies

• Branch Record Buffer
• Call Stack Recorder
52-bit VA/IPA/PAs

• Armv8.2-A introduced support for 52-bit VAs, IPAs and PAs
  • Only available when using 64K granule
  • There are some platforms which are unable to adopt 64K granules

• Armv8.7 extends 52 bit addressing to 4K and 16K granules

• Input addresses:
  • T*SZ field rules relaxed to allow specifying up to 52 bits of address space
  • Can result in an extra level of walk (level -1)

• Output addresses:
  • To enable the larger output addresses, shareability attribute moved from descriptors to TCR_ELx
**PCIe hot-unplug (1)**

PCIe devices can be unplugged at any time

- Could occur in the middle of an access
- PCIe root complex responsible for generating a response after a fixed out
  - Typically 50 ms

If a CPU has outstanding accesses to the removed device, it could be blocked until the timeout expires

- 50ms → 100 million cycles @ 2GHz

Other CPUs can continue to make progress
PCIe hot-unplug (2)

TLBI+DSB waits for all memory transactions using old translation to complete

If a CPU receiving a TLBI simply waits for all outstanding memory transactions to be complete

- A CPU awaiting a PCIe hot-unplugged completion will take ~50mS to acknowledge the TLBI
- CPU that sent out the TLBI now also exposed to a ~50mS delay
- Impact beyond the CPU that is directly talking to the PCIe endpoint
XS attribute

New XS attribute for devices with potentially long delays
• New qualifier to Device-*

TLBI and DSB changes
• Existing TLBIs are unchanged
• New TLBIs added which are only required to wait for transactions with XS=0 (fast) to complete
• Similar changes to DSB

Receiving PE now only needs to track whether outstanding transactions are fast or slow (XS==1)
64-byte atomics

Growing trend for accelerators that support 64-byte atomic load and stores

New store with return value:
- ST64BV Xs, Xt, [Xn|SP]
- ST64BV0 Xs, Xt, [Xn|SP]
  - Substitutes the bottom 32 bits for value in ACCDATA_EL1
- Xs acts a return value

New instructions without return value:
- ST64B Xt, [Xn|SP]
- LD64B Xt, [Xn|SP]

Only permitted to non-cacheable addresses
WFE/WFI with timeout

A lot of potential usage of WFE/WFI is blocked by the unlimited time that the WFx might be asleep

Armv8.7 introduces variants of WFE/WFI with a software specified timeout

• Specified in terms of time, not cycles

WFET Xd / WFIT Xd

• Xd holds a 64-bit value to compare with CNTCVT_EL0
  – Generates a local event to wake the WFxT when CNTCVT_EL0 >= Xd value
  – If WFxT is woken for any reason, the count will be discarded
  – Event only applies to the PE that executes the instruction
Enhancing PAN

- **PSTATE.PAN** was introduced as part of Armv8.1
  - Causes a permission fault for a privileged data access to unprivileged data memory

- Did not consider the case of privileged data access to user execute-only memory
  - AP[1]==0 but UXN==0
  - Siguza’s blog -> [https://siguza.github.io/PAN/](https://siguza.github.io/PAN/)

- Linux had offered the ability to mark memory as User Execute-only to protect JITed code
  - Makes it harder to attack the JITed code, or to find gadgets within it
  - ...But that can be exploited to work around the PAN protection

- **SCTLR_EL1/2** bit to extend PAN so it causes EL1/2 access to fault on pages that have EL0 instruction or data access
  - Permitted to built in any version from v8.1
Future Architecture Technologies
Debug and Visibility

Enhancing software development on Arm
Enabling developers

**Enhanced performance attribution**

*What’s my hot code?*
- Feed in to tools like AutoFDO

*Where in my call stack or call graph?*
- Better attribution of events to real code paths

**Enhanced debug**

Low-impact consumption of call stacks
- Stack unwinding in software is slow and introduces probe effects

Lower-overhead logging
- Capturing call stacks on interesting events, such as malloc/free
Enabling developers

Enhanced performance attribution

What’s my hot code?
• Feed in to tools like AutoFDO

Where in my call stack or call graph?
• Better attribution of events to real code paths

Enhanced debug

Low-impact consumption of call stacks
• Stack unwinding in software is slow and introduces probe effects

Lower-overhead logging
• Capturing call stacks on interesting events, such as malloc/free
Enabling developers

Enhanced performance attribution

What’s my hot code?
- Feed in to tools like AutoFDO

Where in my call stack or call graph?
- Better attribution of events to real code paths

Enhanced debug

Low-impact consumption of call stacks
- Stack unwinding in software is slow and introduces probe effects

Lower-overhead logging
- Capturing call stacks on interesting events, such as malloc/free

ERROR: AddressSanitizer: heap-use-after-free on address 0x60700000dfb5
READ of size 1 at 0x60700000dfb5 thread T0
  #0 0x4007d7 in D(char*) /home/use_after_free.cpp:6
  #1 0x4007d7 in main /home/use_after_free.cpp:45
  #2 0x2b3e6817ac04 in __libc_start_main (/lib64/libc.so.6+0x21c04)
  #3 0x400826 (a.out+0x400826)

0x60700000dfb5 is located 5 bytes inside of 80-byte region previously allocated by thread T0 here:
  #0 0x2b3e669d3b18 in __interceptor_malloc_asan_malloc_linux.cc:62
  #1 0x40079c in C() /home/use_after_free.cpp:18
  #2 0x40079c in B() /home/use_after_free.cpp:35
  #3 0x40079c in A() /home/use_after_free.cpp:39
  #4 0x40079c in main /home/use_after_free.cpp:44
  #5 0x2b3e6817ac04 in __libc_start_main (/lib64/libc.so.6+0x21c04)
Call Stack Recorder Extension (CSRE)

Objective
Capture the call stack in an easy to consume format
• To copy on periodic or event-driven interrupts
• To copy on malloc/free calls

Requirements
Userspace or Kernel controllable
Capture in main memory
• Allows simple `memcpy()` of contents, and scales to capture the full call stack
Low overheads while recording and context switching
Low cost of reading the call stack on malloc/free

Call Stack Record Buffer
• Allocated by kernel or userspace

Call Stack Record
• Value of LR after BL, 8 bytes in size
  • Written on each BL, updating the pointer
  • Faults reported synchronously
• Pointer decremented on each RET

Call Stack Recorder
• Separate controls provided at each of EL0/EL1/EL2, for fast switching of recorder on entry to kernel
• Usual traps for Virtualization
• Separate traps for EL0 register reads and writes
Branch Record Buffer Extension (BRBE)

**Objective**

Capture a recent sequence of branches in an easy-to-consume format
- For statistical capture and feed in to FDO tooling

**Requirements**

Low compute and memory overheads for capture/analysis
- Program trace (ETM) can be expensive in multiple ways
- Prefer uncompressed capture and no need for the program image

Low overheads while recording

Low cost of context switch and on reading

**Branch Record**

- Source VA of taken branch/exception
- Target VA of taken branch/exception
- Info: source/target valid, branch/exception type

**Branch Record Buffer**

- EL1 feature, for recording EL0 and/or EL1
- Accessible via system registers, with 3 per record
  - 96 registers, for up to 32 records
  - Banking system for >32 records
- Invalidate-buffer instruction
  - To prevent future reads revealing old data
- **ISB needed before reading (but not TSB CSYNC)**
More information

https://developer.arm.com/architectures/cpu-architecture/a-profile/exploration-tools

Arm Architecture Reference Manual expected Jan 2021
Thank You
Danke
Merci
Merci
Merci
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
tödö