Compute eXpress Link Status and what's next?

Jonathan Cameron | Huawei

Outline

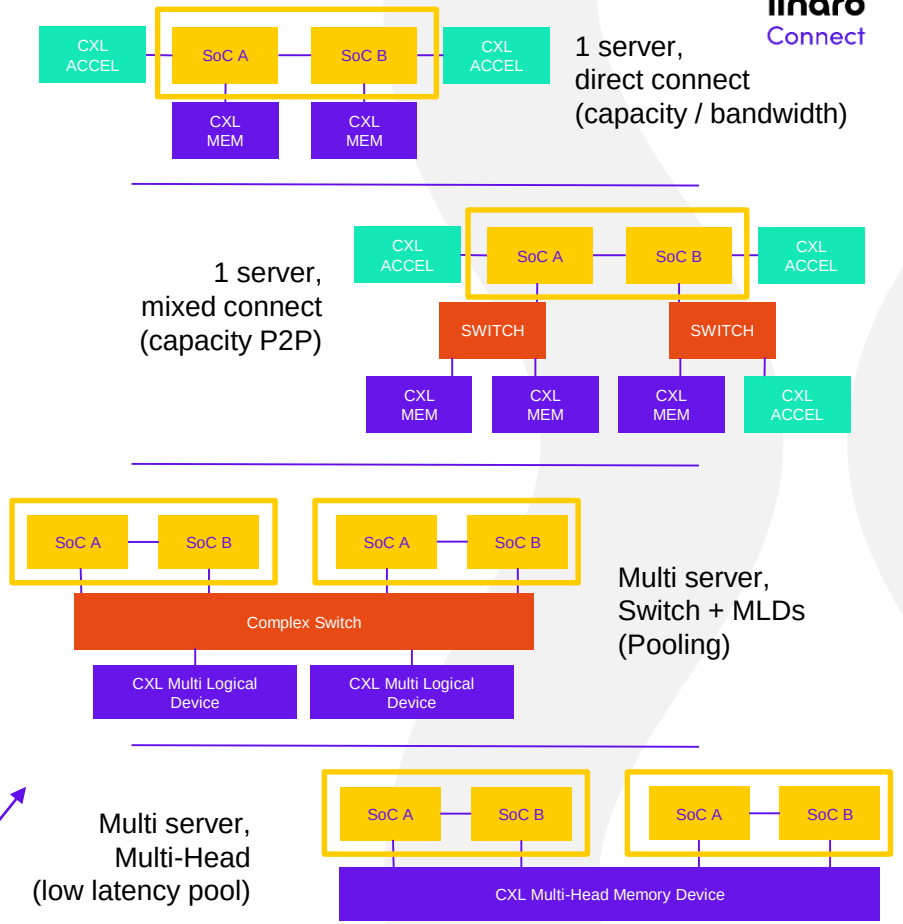
- Brief introduction to CXL
- Memory Expansion
- Architecture specific corners
- Memory Pooling
- Memory Sharing
- What is next?

Notes

- I will go fast.
- I'd rather stop to have a useful discussion than cover all the material.
- Corridor track also available!

What is CXL?

- Interconnect Standard (CXL consortium)
- Closely related to and built on top of PCI Express (gets quicker when PCIe does)
- Enables 3 key things:
  - Low(ish) latency memory access. Main focus of today!
  - Coherent caching of Host Memory by devices.
  - Coherent caching of Device memory by hosts.
- CXL Fabrics (3.0+) enable large scale fabrics. No public enablement yet.



(Eye Candy!)

Key software

linaro Connect

Kernel CXL subsystem

- Healthy multi-vendor community
- Overlaps other subsystems (mm, perf, EDAC)
- Monthly sync call (ask for details)
- LPC uconf.
- linux-cxl@vger.kernel.org

QEMU emulation

- Key platform for ecosystem development.
- Ahead of hardware and kernel.
- Upstream + staging tree on gitlab.
- Arm64 support under review!

Firmware

- Some approaches to CXL are firmware heavy – most early deployments use this.
- Some support proposed for EDK2 /qemu-sbsa.

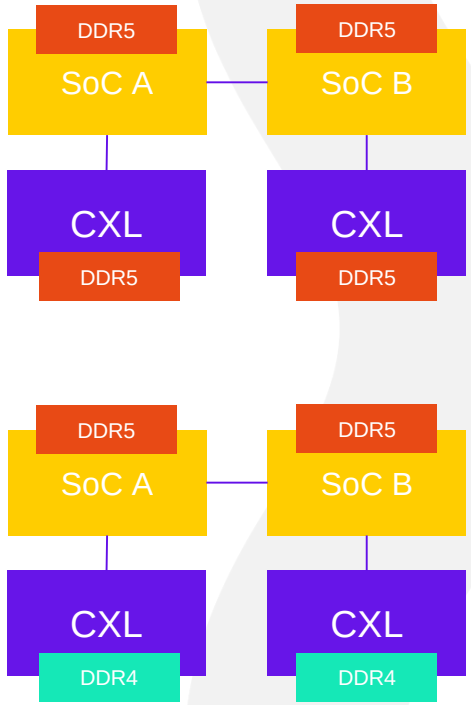
Fabric Management

- CXL provisioning and management is moderately complex.
- Various projects announced, but not clarity yet on which will succeed.
- Challenges in getting test platform.

Memory Expansion

Current Generation hardware

- Expansion of conventional NUMA
- Direct connect for best latency
- (switches enable high fan out)
- Common use cases:
  - Bandwidth Expansion (weighted interleave)
  - Capacity Expansion
  - Technology mixing
- Present at boot / hotpluggable
- Advanced RAS!
- Tiering assistance – Hotness Tracking!



RAS in a CXL world

linaro Connect

Making kernel RAS developers dreams come true!

- OS first control of features. Enabling sophisticated policy decisions.
- FW may not even know devices are there.
- Standardization – some adoption of interfaces for other devices (OCP).

Some examples:

- Rich error reporting, event threshold control etc.
- Memory Scrub control (differentiated reliability!)
- Memory repair (PPR / sparing)

Core RAS features support added to EDAC subsystem. Drivers under review.

- Sysfs driven, policy in RASDaemon etc.

Lots of other RAS challenges from CXL to solve (Isolation / recovery etc)

Architecture specific elements

Arch Corners...

NUMA description for hotplug

- Normal ACPI memory hotplug provides NUMA node as part of event...
- CXL native hotplug does not.
- CXL hotplug relies on CEDT table*
- On x86 NUMA information was retained, not on other architectures.
- Now numa-memblk generalized so everyone can use it
  - Not an arch problem any more!
  - (thanks to Mike Rapoport)
- Simplistic mapping. Will need a revisit in the long term.

Cache flushing by physical address.

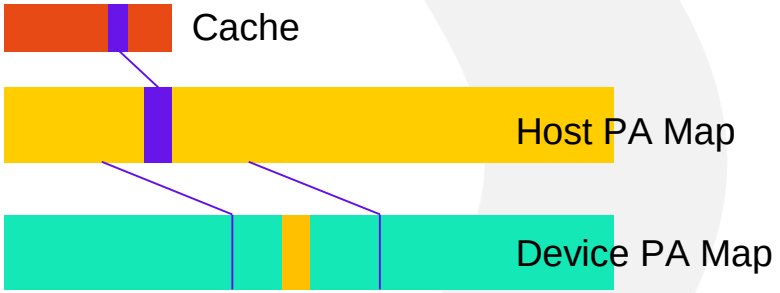
- Why (see next slide)
- X86 has a global cache flush that works (slowly).
- ARM64 has no such instruction.
- System problem – need a generalized solution!

Why do we need a cache flush by PA?

Simple case – we switched the memory.

Time A: Demand fetch, decoder config A

Time B: Demand fetch, decoder config B

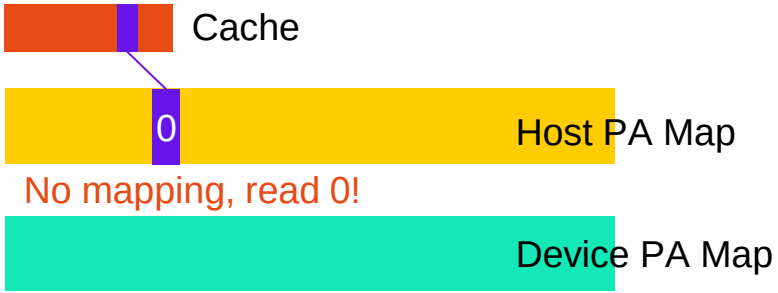


Oops, stale content!

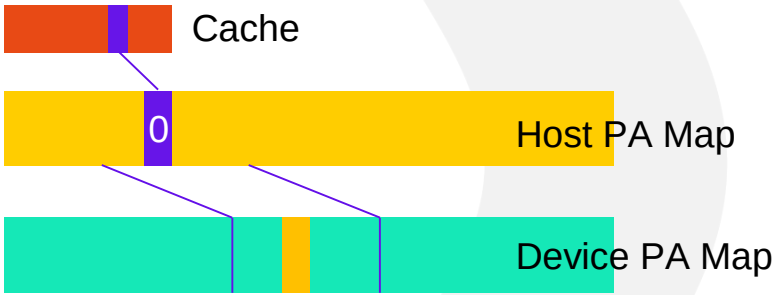
Why do we need a cache flush by PA?

Complex case – we switched the memory and prefetch occurred!.

Time A: Prefetch, decoder not configured!



Time B: Fetch, decoder configured but zeros in cache!



Oops, hallucinated contents!

Cache management subsystem

linaro Connect

Requirements:

- Support various implementations of flush by PA.
- Disaggregated components / interleave.

Minimal Solution

- ‘Device class like’ registration framework.
- Register with arch call back (check name).

Enhancements:

- Flush only the PA range that needs to be (add range flush)
- Trade off full flush (disruptive) vs range flush (may be slow!)

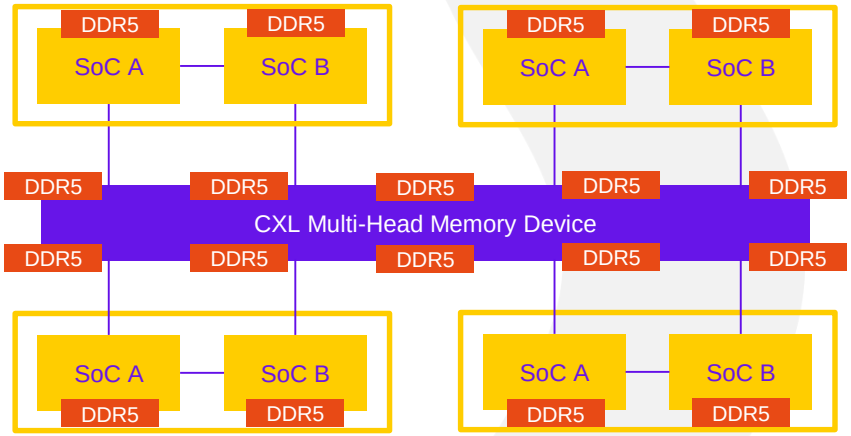
- Driver per implementation?
  - No standards yet…
- Option of an ACPI ‘device’.
  - Example based on (currently dropped) PSCI interface proposal.
  - Worth perusing?

Back to architecture independent

Memory Pooling

Next generation hardware

- Solution to ‘stranded memory’ / VM packing challenges.
- Unified pool of memory accessible to small number of servers (< 32)
- Multiheaded Devices (low latency)
- Dynamic allocations to medium term activities (VM, specific applications – e.g. DB, AI).
- Kernel stack available, but not merged.



8 head pool – dual host connections

Memory Pooling – Challenges

Challenges:

- New management concepts – wire up to cloud orchestrator.
- Memory for 1 purpose – once done, give it back!
- Is DAX the way to go?
- How to work with CoCo?
- Application specific memory virtualization.
    - CXL emulation?
    - Virt-IO solution?

Memory sharing

linaro Connect

Why?

- Data duplication / sharding.
  - RocksDB / Apache Arrow etc.
  - **Read only (or mostly)**
- Low latency data transfer.
- Zero copy live migration?
  - Pipe dream or real? (long way to go!)

How to use it?

- Consistent presentation / DAX?
- File abstractions / FAMFS.
- Stack specific memory

Software managed coherency

- If only coherency was cheap.
  - Extra bus transactions needed.
  - Tracking hardware on device.
- Seems it isn't…
- Need to handle lack of coherency between hosts.
  - Clean write backs! (who would do that?)
  - How does an application flush writes?

More... architecture specific elements

Application driven interhost cache flushing?

linaro Connect

Why?

- Producer / consumer model
- Synchronization out of band (so ignore that)
- What does Point of Coherence mean in a multihost system?
- Do we need a Point of Deep Coherence? ☺

Simple solutions (maybe?)

- For memory in a CXL fixed memory window (PA range) PoC is in remote memory. Works, but…
- Temporal sharing. May only be visible to this host for now – so why flush to remote memory?
- Need to distinguish shareable types:
  - Shared
  - May be shared later but needs explicit flush before that!

Wrap up. I'm ~~probably~~ out of time...

Getting involved...

Scratch your own itch.

- What applications do you care about?
- Can we test them on QEMU?
  - Emulation is pretty easy to do!
- Can we map them to available hardware?

There are pitfalls that can be avoided...

- Implementations out there don't always do what Linux expects...
  - Low memory hole,
  - SPA concept on AMD,
  - Hardware local DDR / CXL interleave.

Big gaps...

- Firmware.
  - We don't need to do the FW first flows, but without them testing is limited.
- Full open source fabric management stacks for pooling.
  - Redfish,
  - OpenBMC,
  - Generic cloud orchestrator interactions.

Thank You!