OPENFABRICS ALLIANCE

14th ANNUAL WORKSHOP 2018

New types of Memory, their support in Linux and how to use them with RDMA

Christoph Lameter, Ph.D., R&D Team Lead

Jump Trading LLC

April 5, 2018

Overview

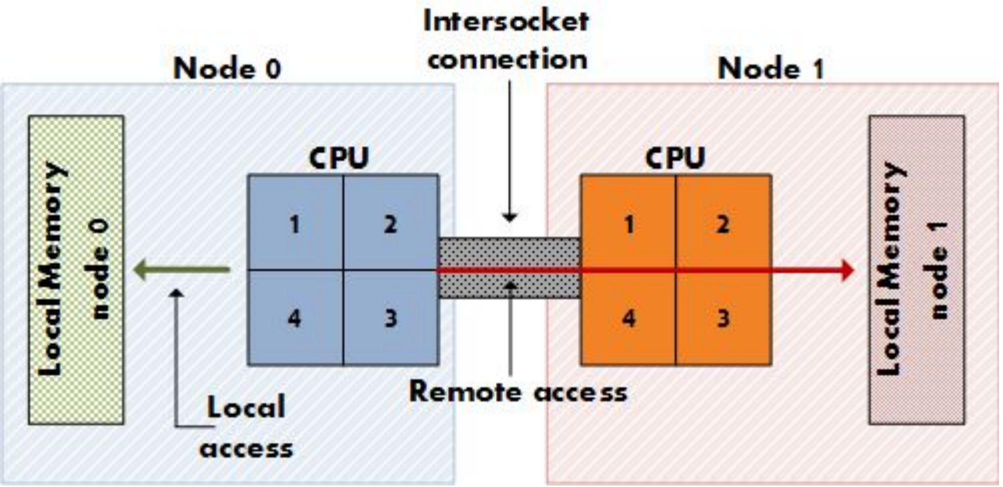
- ***Why talk about memory?***
- ***Brief review of traditional memory support in Linux***
- ***New memory features added to Linux in the last years***
- ***Upcoming work and future memory support***
- ***Questions?***

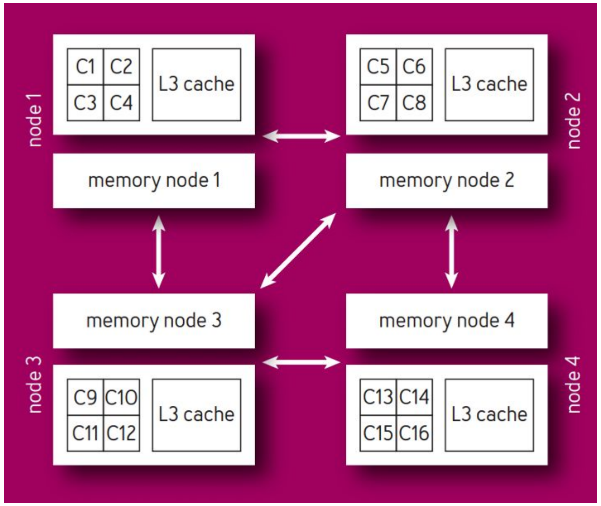
OPENFABRICS ALLIANCE

A Brief Overview of Traditional Memory in Linux

“Classic” Memory handling

- Memory with uniform access.
- Memory Zones (f.e. for DMA)
- NUMA Non Uniform Memory Access
- Memory Affinities
- Memory Policies
- Cpusets, cgroups and containers
- Manual process migration
- RDMA
  - Possible to all of these types of memory.
  - Memory is pinned when registered for RDMA
  - Becomes unmovable.
  - Registering large areas of memory can lead to kernel failures





Features added in the last years to Linux Memory management

New Features and advanced memory support

- **New Memory Zones**
  - Movable memory
  - Device Memory (ZONE_DEVICE)
- **Innovative use of NUMA nodes**
  - HBM memory node
  - Memory node
  - Memoryless node
  - Device Memory Node
- **Advanced memory support**
  - AutoNUMA: Automatic memory migration
  - THP: Transparent Huge Pages
  - On demand paging (ODP) for RDMA
    - Avoids pinning
    - Can register large amounts of memory (f.e. the whole process address space)
    - Memory registration can be moved out of critical loops.

Special NUMA node use cases

- **HBM support**
  - HBM in use for GPUs etc
  - High bandwidth but also high latency
  - Supported in the Xeon Phi f.e. as a separate NUMA node.
  - Memory may behave differently under RDMA since the interface to HBM does not follow the classic memory channel design.
- **NUMA nodes with processors only**
  - Special use case where a load is computational intensive.
  - All memory access is remote.
  - Only makes sense for extremely high processor dependent loads.
- **NUMA nodes with memory only**
  - Expand memory to keep large datasets continually accessible without having to access slow secondary media.

Nonvolatile Memory

- **Memory contents are preserved**
- **NVM can be addressed like regular memory**
- **New challenges with data consistency**

In a typical use case NVM is handled like a regular block device.

I.e. (output of ls)

```
ls -l /dev/pmem0
brw-rw---- 1 root disk 259, 3 Mar 29 14:03 /dev/pmem0
```

It can be formatted and mounted

```
mkfs.ext4 /dev/pmem0
mount -o dax /dev/pmem0 /nvdimm
```

And files can be stored and retrieved from it like any other disk.

DAX (Direct Access to Device Memory)

File contents of a DAX file does not use the page cache for mmapped accesses.

Since the Nonvolatile Memory is directly accessible: Mmap directly establishes references to the data on the NVRAM module.

Contents cannot be evicted from memory thus there is no need to use the page cache and subject pages to eviction.

The 4K size restriction of pages in the page cache is not valid for DAX files. 2M and 1G mappings can exist in an NVM file. This significantly reduces OS overhead and TLB pressure.

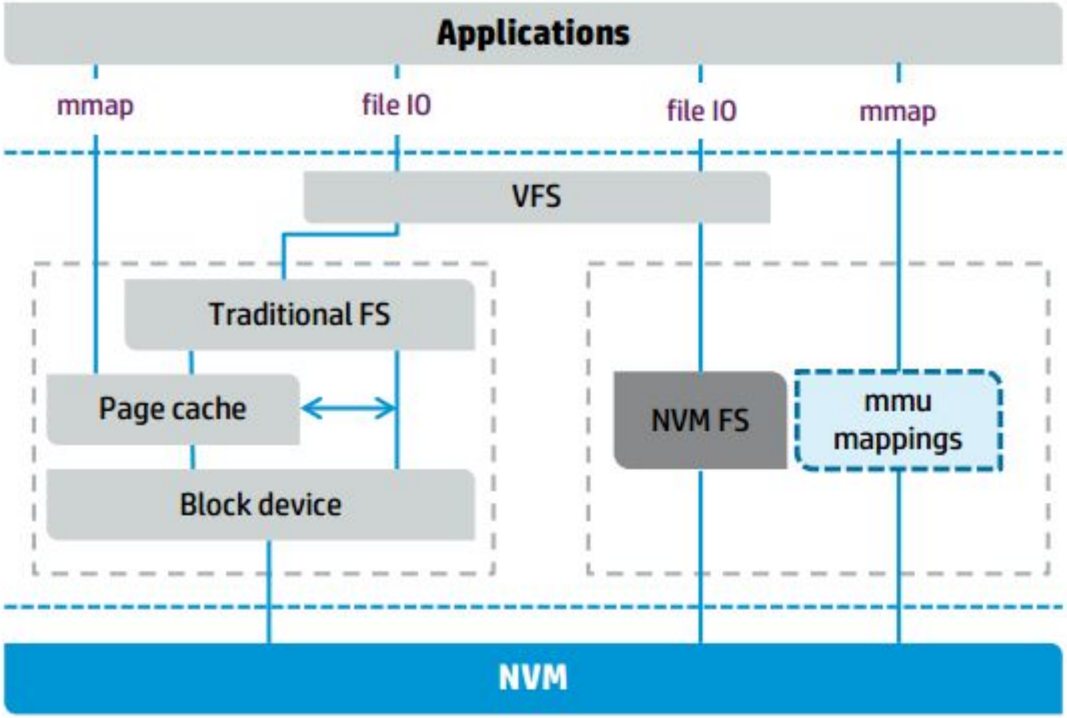
NVM challenges

Filesystems have something like transactions to ensure that the filesystem is always coherent. So we need something like that also on the level of NVM accesses. For that the processors cache coherency handling needs to be modified to allow transactions on NVM. Cachelines needs to be written back to NVM memory in such a way that a consistent state exists in case of failure.

If the access to the NVM volume is through a filesystem then the filesystem coherency mechanisms can ensure file consistency within the constraints of each of their filesystems.

NVM design

- Block Devices
- Additional direct DAX path via MMU
- Filesystem can decide which path is allowed
- Block device can control flushing
- Preserve POSIX file system semantics as much as possible

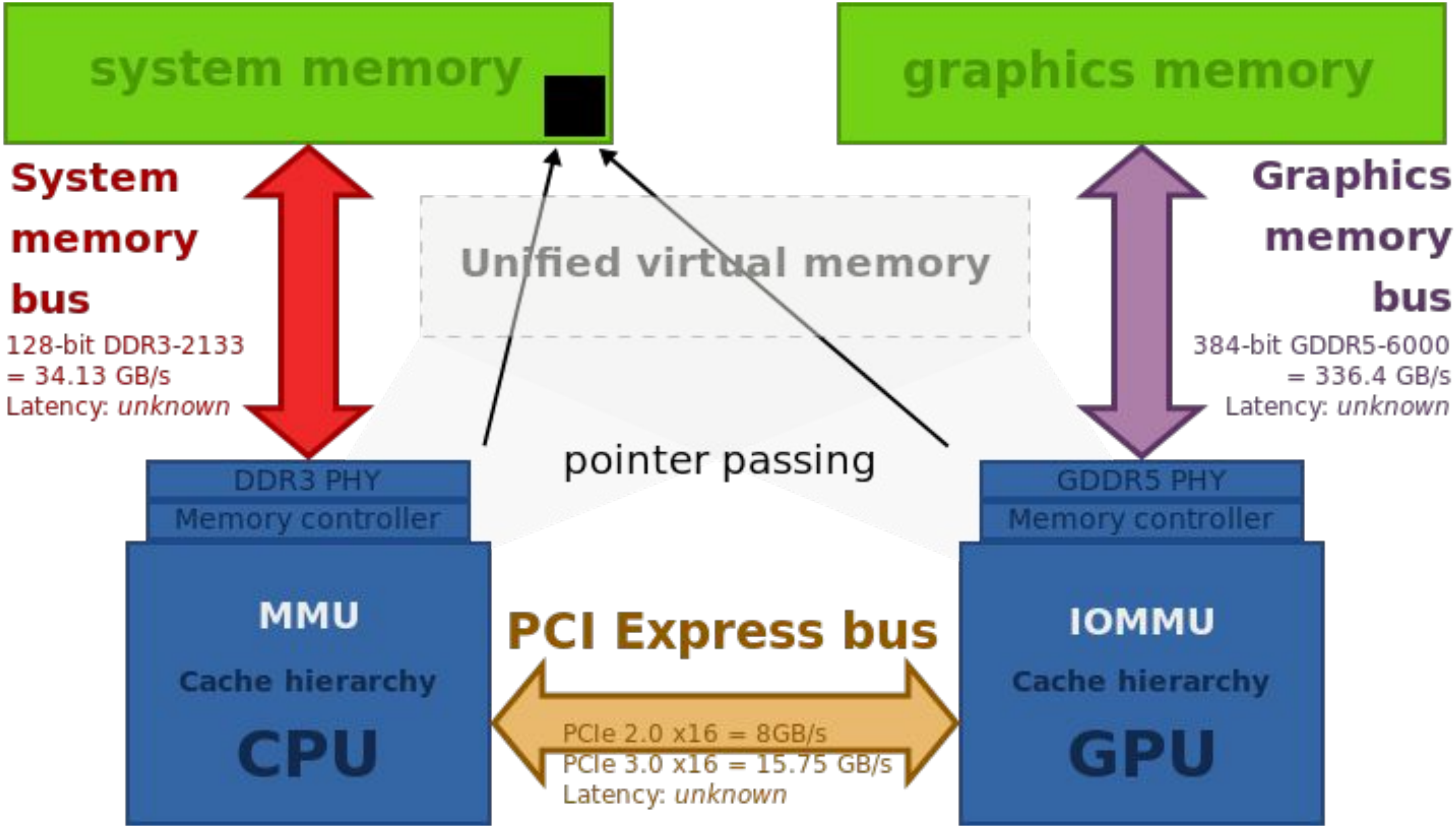


Non RDMA-able Non volatile memory

Linux supports two different types of Non volatile memory. The overhead of keeping state per 4K page is about 2% of the capacity of a NVM device. NVM device have huge capacity and so we may end up dedicating significant amount of regular memory for NVM management.

ZONE_DEVICE has to be available for an NVM device in order to have metadata for 4K pages. If not the operations on NVM memory areas are limited.

Heterogeneous Memory



Heterogeneous Memory

- Enhancement of the mmu_notifiers build on the cross platform HSA (Heterogeneous System Architecture) standard.
- Managing large memory for I/O device (GPUs, coprocessors) with a virtual address space that allows the use of cross device pointers.
- External memory behind PCI-E and other device specific barriers that may be high latency and have their own coherency methods.
- HMM synchronizes memory views on both sides
- ZONE_DEVICE used to manage reference to memory that does not exist on the host side
- RDMA issues: Pages may never be mapped into main memory. Significant overhead if pages have to be copied for I/O. Maybe better to use PeerDirect?

Future types of Memory

Future

- **RDMA squared**
  - Universal RDMAbility via a system wide ODP-like scheme?
  - Shared distributed memory via RDMA (shmem style) and process sharing
  - Remote "Persistent" Memory pools?
- **Remote RDMA for large storage arrays**
  - pNFS with RDMA layout. Access large array of NVMe memory via RDMA.
  - PeerDirect in remote nodes. Substitute device access for memory access.
  - NVM memory pools. Shared shared memory pools and filesystems that support such a shared memory pool.
- **Ultimately we need a multilayered generic memory infrastructure that moves data over a variety of busses to allow for optimal latency.**

OPENFABRICS ALLIANCE

14th ANNUAL WORKSHOP 2018

Questions ?

Christoph Lameter

cl@linux.com

Notes

HBM See https://arxiv.org/pdf/1704.08273.pdf
SNIA https://www.snia.org/sites/default/files/technical_work/final/NVMProgrammingModel_v1.2.pdf
Wiki on pmem http://pmem.io
https://github.com/pmem