Non CPU threads/process lifetime

- Non CPU thread (like GPU thread) always depend on CPU one ?
- Lifetime bind to mm struct, should we bind to task ?
- Do we care about having non CPU process ? I think not

GPU threads

- With 10000, 100 000 or more thread on GPU we do not want per thread structure or tracking
- Thread form group of thread (often 32 or 64 threads in a group)
- All thread in a group run concurrently (single instruction decoder)
- For 32/64 Cores there is often 4/8/16 group of thread in flight so on average there is a runnable group
- Group execute interleaved (like CPU HT)

GPU threads scheduling

- GPU 10000,100000 threads or more
- Preemption extremely costly
- Rush to finish thread groups better option
- Some hw can have multiple process in flight and slowly prioritize one over other
- Thread are manage in silicon, no software scheduler
- GPU driver can stop|pause thread
- Software control is mostly about giving a priority
- Thread are spawn through job queue (ring buffer containing cmd usualy)

GPU threads scheduling

- Common property is a priority value
- Other metric people would like to see ?

Sharing address space

- Pointer means same thing on CPU and device
- Crucial for “complex” data structure (list, tree, …)
- Memory model might be different
- Cache coherency mandatory for share memory
- Device memory might not be accessible by CPU

Sharing address space (ATS/PASID)

- ATS/PASID PCIE extension where IOMMU walk CPU page table and allow device to directly access virtual address of a process
- IOMMU use PASID (process address space ID) 16bit id device use to select proper process
- PASID and process binding in kernel space
- Can we have unique ID for lifetime of mm ?
- Probably not because hw can implement less then 16bit for ID and some hw out there only have 8bit
- Dynamic binding of ID ?
- ARM require static ID

Device memory

- Bandwidth from 256GB/s to 1TB/s more coming
- Only GPU/FPGA/DSP can saturate such thing
- System memory bandwidth (DDR3/DDR4/...) does not seem to catch-up
- PCIE limitation on what CPU can access and do (no CPU atomic operation on PCIE bar)
- PCIE latency

New bus for device memory (unofficial and unsanctioned)

- AMD Coherent Fabric to replace PCIE and allow coherency for all devices (CPU included) with GPU memory
- IBM CAPI (Coherent Accelerator Processor Interface) (v1 sit on top of PCIE)
- NVidia Nvlink same idea as AMD but depends on platform to implement it
- Intel ? Unknown

CPU un-accessible device memory

- Can not wait for new bus
- Have to support in some way
- HMM (Heterogeneous memory management) ?

HMM and address space mirroring

- Helper to maintain a device specific shadow table of CPU page table (synchronizing both)
- Support migrating anonymous memory (unvisible from CPU but pagefault trigger migration back)
- Experimental share memory migration
- WIP page write protection for device atomic operation (work around limited PCIE atomic semantic)
- Try to work with several hardware (Mellanox, NVidia so far)

CPU accessible device memory

- Regular struct page for device memory ?
- Similar problematic as for persistent memory
- Allocation to device memory should be explicit and never happen for “regular” page allocation (ie device driver knows best)
- Too many places expect struct page, should we need to change that (Jan Kara cleanup about vma handling first second something like Dan Williams pfn_t for DAX) ?