Here’s an exhaustive list of topics that can be asked in a **GPU Kernel Senior Engineer Interview**, especially in **Linux Kernel GPU driver development**. I'll categorize them for you:

**1. GPU Architecture and Fundamentals**

* **GPU vs CPU architecture** (SIMD, SIMT, Threads, Warps, Cores, etc.)
* **GPU pipeline**: Vertex shader, fragment shader, compute shader, rasterizer, texture unit, etc.
* **Memory hierarchy in GPU**: L1, L2 cache, global memory, shared memory, texture memory, etc.
* **MMU in GPU**: IOMMU vs SMMU vs GMMU (GPU Memory Management Unit)
* **GPU Virtualization**: SR-IOV, VFIO, GPU passthrough, mediated devices (mdev)
* **Workloads and scheduling**: Compute workload vs graphics workload, scheduling policies
* **CUDA/OpenCL basics**: Thread blocks, warps, grid, streams, etc.
* **GPGPU programming basics**: General-purpose GPU usage, parallel programming

**2. Linux Kernel GPU Driver Internals**

* **DRM (Direct Rendering Manager)** architecture in Linux
* **KMS (Kernel Mode Setting)**: What is it? Why is it needed?
* **Framebuffer device (fbdev)** and its interaction with DRM/KMS
* **Device Tree binding for GPU** in ARM-based SoC
* **Probe function in GPU driver**: What happens when a GPU driver loads?
* **Hot plug handling in GPU**: How GPU devices are enumerated and initialized during hotplug
* **PCIe device enumeration for GPU**: Role of PCIe bus, BARs, MMIO, DMA, etc.
* **PCIe link training**: L0, L0s, ASPM, L1 substates, etc.
* **GPU power management**: Runtime PM, suspend/resume, dynamic power gating
* **GPU device memory allocation**: CMA (Contiguous Memory Allocator), ION, DMA-buf, etc.
* **GPU buffer sharing**: dmabuf, prime, fence mechanisms

**3. Memory Management in GPU Drivers**

* **IOMMU/SMMU usage in GPU drivers**
* **Memory mapping**: mmap(), remap\_pfn\_range(), dma\_map\_single()
* **DMA API in kernel for GPU**: Coherent vs streaming DMA
* **GPU VRAM management**: GEM (Graphics Execution Manager) vs TTM (Translation Table Manager)
* **Unified memory in GPU**: Heterogeneous memory architecture (HMM)

**4. Linux Kernel Internal Interactions**

* **Interrupt Handling for GPU**: IRQ handler, bottom halves, threaded interrupts
* **Polling vs Interrupt-driven IO** in GPU context
* **Firmware interaction with GPU**: Loading firmware during driver probe
* **Power domains and regulators for GPU**: Dynamic voltage and frequency scaling (DVFS)
* **Thermal management in GPU**: Cooling, thermal sensors, thermal throttling

**5. GPU-specific Linux Subsystems**

* **DRM subsystem API flow**: drm\_device, drm\_driver, drm\_crtc, drm\_plane, drm\_connector
* **Modesetting in GPU**: Atomic modesetting, atomic state updates
* **Display Pipeline Flow**: From framebuffer to HDMI/DP output
* **Color formats and color spaces**: YCbCr, RGB, planar, packed formats
* **Linux Kernel display drivers**: DisplayPort, HDMI, eDP, MIPI DSI
* **GPU hotplug handling**: Runtime power management, suspend/resume

**6. Linux Kernel Call Flows for GPU**

* **Driver probe sequence**: PCIe enumeration → Device registration → Probe function → DRM registration
* **Framebuffer creation**: devm\_kzalloc → mmap → dma\_map\_single
* **Rendering pipeline**: User space (Xorg/Wayland) → DRM → Framebuffer → Display
* **GPU reset flow**: Soft reset, hard reset, runtime PM

**7. PCIe Fundamentals for GPU**

* **PCIe enumeration process for GPU**
* **BAR (Base Address Register)**: BAR0, BAR1 usage in GPU
* **Memory Space and IO Space in PCIe**
* **MSI/MSI-X interrupts in GPU**
* **PCIe hotplug and GPU**
* **Host memory vs Device memory mapping**
* **DMA from GPU to Host**: DMA scatter-gather, IOMMU mapping

**8. Virtualization and GPU Passthrough**

* **VFIO (Virtual Function IO) for GPU passthrough**
* **SR-IOV (Single Root IO Virtualization) in GPU**
* **mdev (mediated device) concept for GPU sharing**
* **KVM-QEMU passthrough for GPU**

**9. Kernel Debugging and Tracing for GPU**

* **ftrace for GPU driver call flow tracing**
* **perf for GPU performance analysis**
* **dmesg and printk for GPU kernel logs**
* **Debugfs usage in GPU drivers**
* **sysfs interaction for GPU power management**

**10. GPU Driver Code Flow (Critical)**

* **PCIe probe function flow for GPU**
* **Device Tree parsing in GPU drivers**
* **Linux Kernel mode setting (KMS)** flow
* **drm\_gem\_mmap vs remap\_pfn\_range**
* **DMA-buf fencing** and GPU buffer sharing
* **Runtime power management flow for GPU**
* **Reset flow of GPU in case of failure**

**11. Advanced Topics (Bonus Edge in Interview)**

* **Unified Memory Access (UMA)** in GPU
* **Asynchronous GPU reset and recovery**
* **OpenGL/DirectX/Vulkan driver architecture**
* **Page flipping, Vsync, and triple buffering**
* **GPU MMU (Memory Management Unit) handling**
* **Context switching in GPU** (process context to kernel context)
* **Linux DRM driver’s atomic commit API**

I'll analyze our past conversations to identify which of these GPU topics you've already covered and which ones are remaining. One moment. Based on our past conversations, here's the analysis:

**✅ Topics You Already Know Well**

1. **PCIe Architecture for GPU**:
   * PCIe enumeration, BAR, MSI/MSI-X, hotplug
   * DMA, memory mapping, host vs device memory
   * PCIe link training and power management
2. **Memory Management in Kernel for GPU**:
   * IOMMU/SMMU (extensively discussed)
   * DMA coherent vs streaming, scatter-gather
   * mmap(), remap\_pfn\_range(), dma\_map\_single()
3. **Interrupt Handling**:
   * MSI, MSI-X, IRQ handling in PCIe devices
   * Bottom halves, threaded IRQ for GPU
   * PCIe hotplug and reset flows
4. **Linux Kernel Subsystems**:
   * Platform drivers, PCIe bus drivers, device tree usage
   * Dynamic power management (runtime PM, DVFS)
   * Kernel boot process and device enumeration
5. **VFIO/SR-IOV Virtualization**:
   * PCIe passthrough using VFIO
   * mdev (mediated devices)
   * SR-IOV in PCIe devices

**🚧 Partially Covered Topics (Need Deep Dive)**

1. **GPU Pipeline**:
   * You haven't explored the full graphics pipeline (Vertex shader → Fragment shader → Rasterizer → Display).
   * You lack knowledge of compute shaders and general-purpose GPU (GPGPU) workloads.
2. **Display Stack in Kernel**:
   * **DRM/KMS (Direct Rendering Manager / Kernel Mode Setting)** is untouched.
   * You haven't explored framebuffer management, atomic commit, vsync, and triple buffering.
   * Color formats like YCbCr, RGB, planar formats — partially discussed.
3. **GPU Power Management**:
   * You covered runtime PM, but not dynamic voltage and frequency scaling (DVFS).
   * No discussion on thermal throttling, GPU clock gating, and power domains.
4. **GPU Buffer Sharing and Synchronization**:
   * You haven’t covered dmabuf, fence mechanisms, or buffer sharing between devices.
   * No knowledge of prime buffers in GPU.
5. **Firmware and Microcode for GPU**:
   * You haven't explored how firmware loads during GPU boot-up.
   * No knowledge of GPU microcode updates and firmware blobs.
6. **Reset and Recovery in GPU**:
   * Soft reset, hard reset, device recovery after a crash — untouched.
   * No knowledge of GPU runtime reset flows.

**❌ Completely Untouched Topics (Critical Gaps)**

1. **DRM/KMS Driver Architecture** (highly important):
   * What is drm\_device, drm\_driver, drm\_plane, drm\_connector, drm\_crtc?
   * How does atomic commit work?
   * How does framebuffer connect to HDMI/DP output?
   * What is page flipping and vsync?
2. **GEM vs TTM Memory Management**:
   * GEM (Graphics Execution Manager)
   * TTM (Translation Table Manager)
   * Unified memory architecture (UMA)
3. **GPU Reset and Error Recovery Flow**:
   * Kernel flow when GPU crashes or encounters errors.
   * Recovery process and state machine.
4. **Kernel Call Flow for Display Pipeline**:
   * Call flow from open("/dev/fb0") → mmap() → remap\_pfn\_range() → display.
   * How framebuffers, display controllers, and pixel pipelines work.
5. **Debugging GPU Kernel Driver**:
   * Using perf, ftrace, debugfs to trace GPU kernel calls.
   * Debugging kernel crashes or hung GPU devices.
6. **Advanced Power Management**:
   * Dynamic voltage and frequency scaling (DVFS)
   * Power gating, clock gating in GPU
   * Thermal management and throttling
7. **OpenGL/Vulkan/DirectX Integration**:
   * How user-space API (like OpenGL) talks to GPU drivers.
   * IOCTL flow from user space to kernel driver.

**✅ What Should You Do Now?**

Given that you’ve already covered the PCIe and memory management parts deeply, your fastest path to cracking **GPU Kernel Senior Engineer** roles is:

**Priority 1 (High ROI)**

1. **DRM/KMS Driver Architecture** (most critical).
2. **Framebuffer to HDMI/DP pipeline**.
3. **GEM/TTM memory management**.
4. **Power management and thermal throttling**.

**Priority 2 (Interview Edge)**

1. **Firmware loading and microcode updates**.
2. **GPU crash recovery and reset flows**.
3. **Buffer sharing and synchronization using dma-buf/fence**.