Porting Nouveau to Tegra K1

How NVIDIA became a Nouveau contributor

Alexandre Courbot, NVIDIA
FOSDEM 2015

The Story So Far...

In 2014 NVIDIA released the Tegra K1 SoC

- 32 bit quad-core or 64-bit dual core ARM
- 192-cores low-power Kepler GPU (OpenGL 4.3, CUDA)
- Desktop Kepler already supported by Nouveau

2014/02/01: NVIDIA to contribute Nouveau GK20A support

(Incomplete) Credits

NVIDIANs:

Thierry Reding
Terje Bergström
Gregory Roth
Vince Hsu
Ken Adams
Lauri Peltonen
Stephen Warren
Mark Zhang

… and the whole Nouveau community!

NVIDIA.

Outline

GK20A/Nouveau overview

Nouveau bringup on Tegra K1

Challenges with memory management

Engines layout on Tegra

User-space (Mesa)

NVIDIA

GK20A Overview

Fully-featured Kepler part with unified shaders and per-process virtualization of the GPU

- Each process gets its own GPU context
- Memory virtualized per-context
- Graphics jobs submitted by user-space using pushbuffers

NVIDIA

Nouveau Architecture

Supports GPUs from Riva TNT (1998) to Maxwell (2014)

- Extremely modular
- GPU literally an assembling of engines and sub-devices

Supporting GK20A means

- Finding/writing engines/subdevs for the chip
- Allowing Nouveau to run on Tegra

NVIDIA.

Platform Bus Support

Nouveau expects the GPU to be on a PCI bus

- Provides GPU registers & BARs I/O addresses
- `pci_map_page()` used to map system RAM to GPU

Abstract the bus and add platform bus support

- I/O addresses provided by Device Tree
- Replace deprecated `pci_map_page()` with DMA API

→Nouveau can be instantiated from PCI or Device Tree

NVIDIA.

No VBIOS

Video BIOS provides useful information (e.g. voltage tables for DVFS) and also performs critical initialization

- Alternate way to provide power information via per-chip static tables
- Perform necessary initialization for GK20A in-driver

NVIDIA.

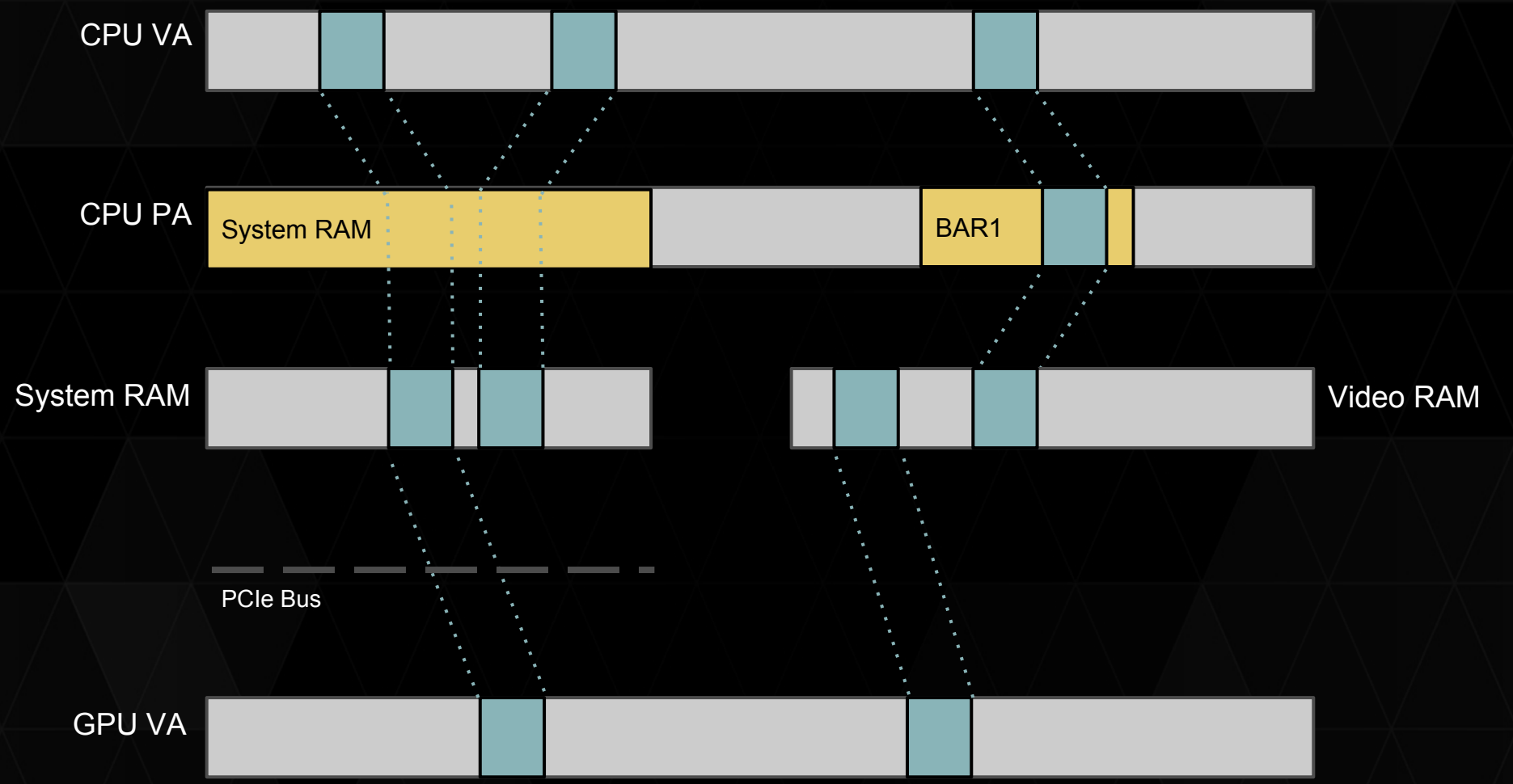
No VRAM

GK20A has no video memory of its own

- GPU is a direct client of Tegra's Memory Controller
- Free and direct access to system memory
- Huge consequences for the driver

NVIDIA.

Address Translation on Desktop Kepler



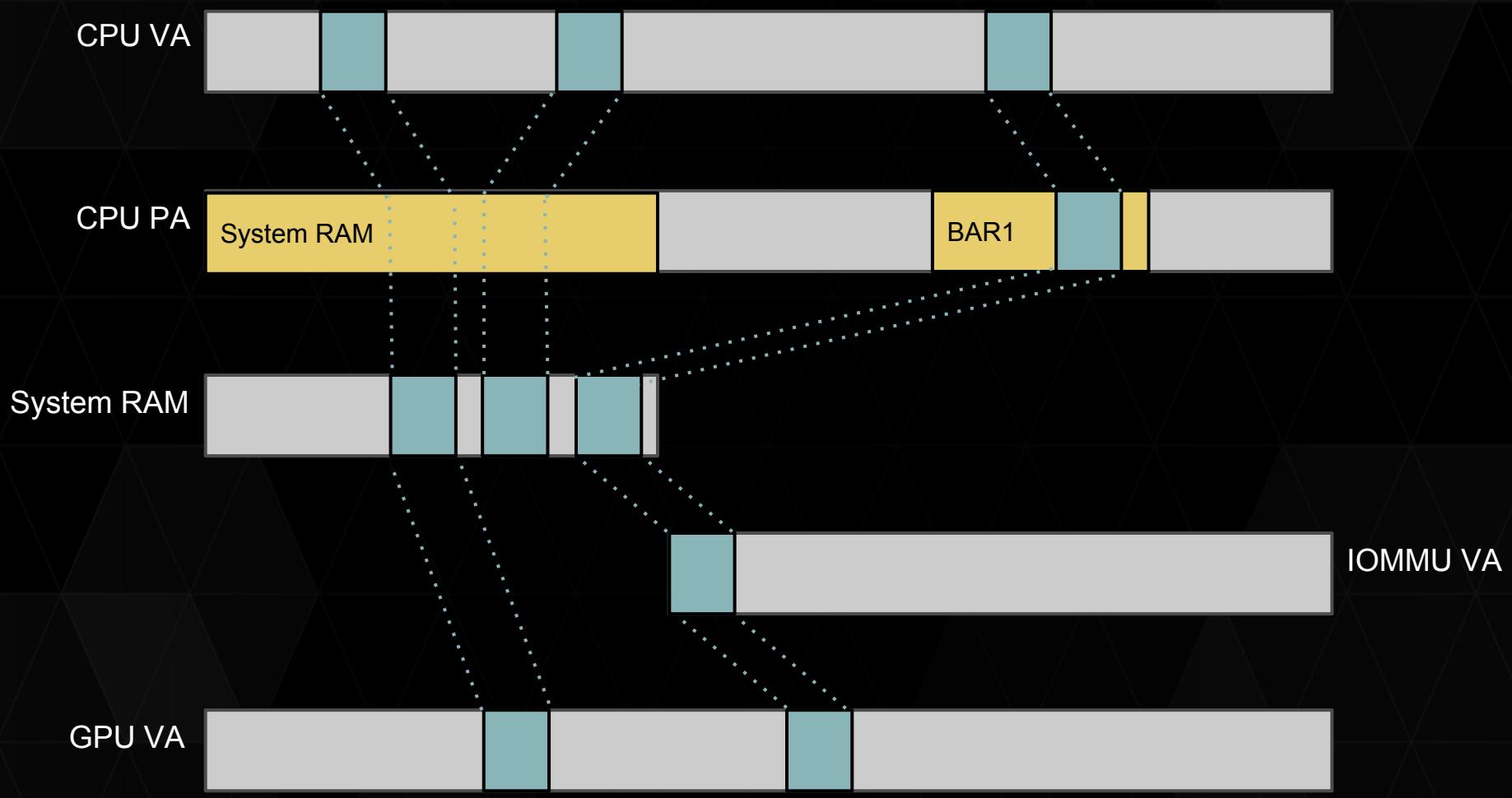
NVIDIA

Nouveau Memory Model

- 2 allocation targets:
  - VRAM
  - TT (system memory mapped to GPU)
- Target specified at buffer creation time
- Coherency maintained thanks to BAR1 (for VRAM) and PCIe (for TT)

NVIDIA.

Address Translation on Mobile Kepler



NVIDIA

Mobile Kepler Memory Model

- No more dedicated video memory
- All allocations in system memory
  - Not a carve out!
- No coherency between CPU and GPU
  - Must flush/invalidate CPU cache ourselves

NVIDIA

Living Without VRAM

How to handle VRAM allocations?

- Emulate VRAM?
  - Sub-optimal memory management
- Dismiss VRAM allocations altogether?
  - Requires more changes in the kernel & Mesa

Decision taken to not use a RAM device for GK20A

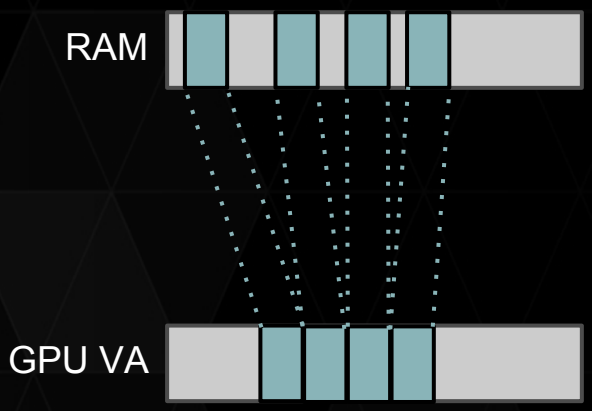
- Better reflects reality, simplifies memory management
- User-space needs to be aware of no-VRAM devices

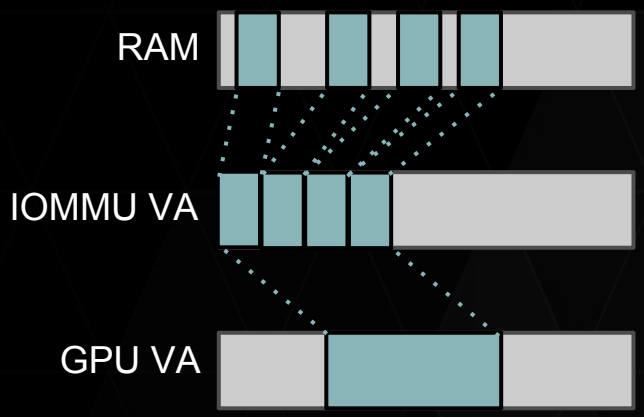
NVIDIA.

Using IOMMU

IOMMU introduces a second level of address translation

- Useful to “flatten” context objects
  - Instance blocks, PGTs, etc.
- Also allows to maximize large page usage on the GPU
  - IOMMU more efficient than GPU MMU





NVIDIA

CPU/GPU Coherency

- Handled transparently by PCIe for desktop
- No such thing on Tegra: explicitly flush/invalidate buffer objects (DMA API)
- New flag for objects that must always be coherent
  - Fences, GPFIFOs
- ARM makes things more difficult
  - A memory page cannot be mapped twice with different attributes
  - Kernel already maps lowmem (first 760MB) cached
  - Cannot remap this memory with uncached attribute

NVIDIA

Multiple CPU Mappings Coherency

How to address the coherency issue?

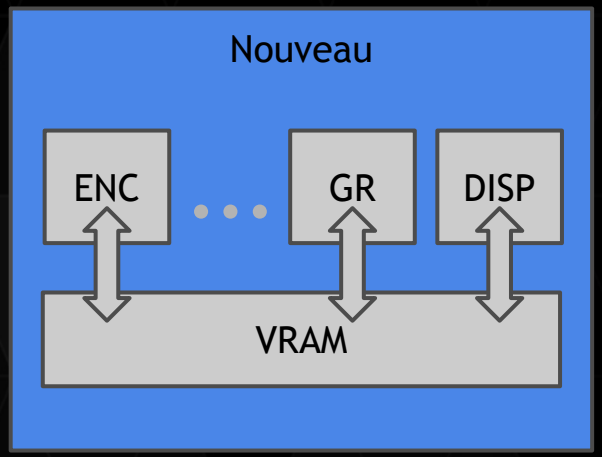
- Use GPU path when writing coherent buffers
  - PRAMIN window (slow)
  - BAR1 (relatively scarce resource)
- Allocate coherent buffers using DMA API
  - dma_alloc_coherent() can fix the lowmem mapping
  - end up with permanent kernel mapping

NVIDIA.

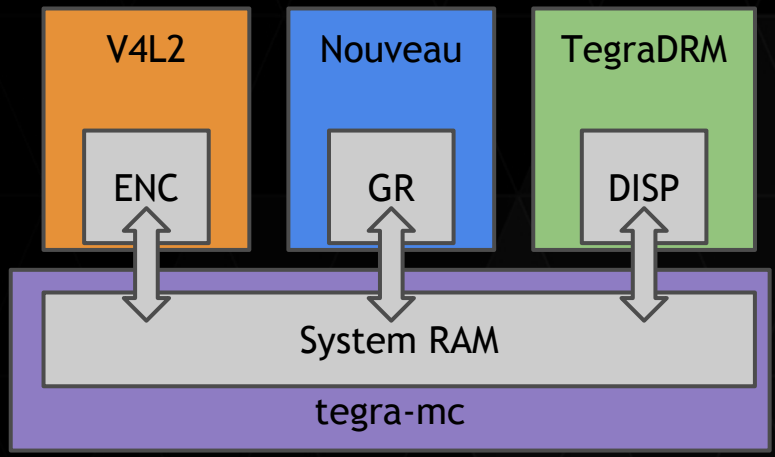
Engines Layout

- GeForce GTX 680 (GK104) provides a graphics engine (GR), display controllers, 3 copy engines, video decoder, video encoder, VRAM, ...
- GK20A only includes a graphics engine
  - Other functions already provided by different Tegra IPs

Discrete GPU



Tegra



NVIDIA

Engines Layout

PRIME support is critical for this setup

- Export required to display GPU buffers

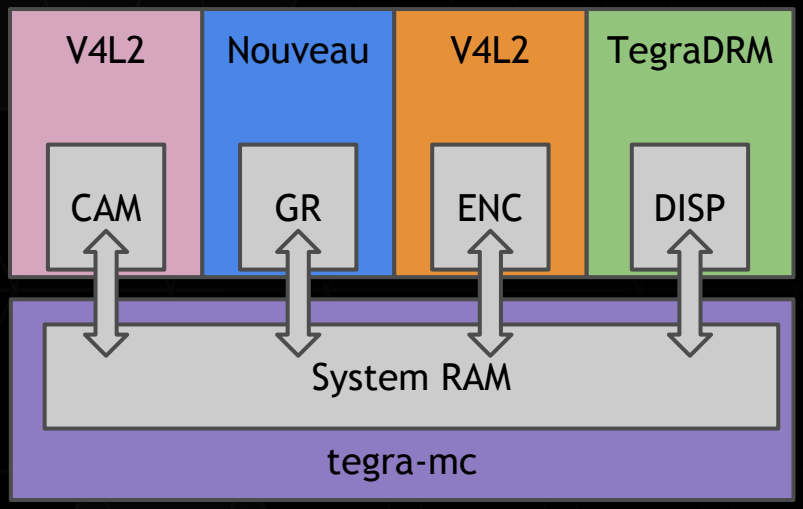
Tegra K1 perfect fit for render-nodes

- `card0` (tegradrm) is the display device
- `renderD128` (nouveau) is the render device

→ requires support at application or Mesa level

NVIDIA.

Who Should Provide Memory?



The first driver in the chain?

A neutral allocator? (e.g. ION)

Why should each driver have its own allocator?

How to handle different engines capabilities?

NVIDIA

User-space (Mesa) changes

~25 LoC changed to recognize GK20A
… and Mesa fully works

Some work required to avoid VRAM allocations

Some more work to integrate seamlessly with tegradrm?

NVIDIA

Conclusion

GK20A close to work out-of-the-box with Nouveau

Remaining tasks:

- Firmware distribution
- A few more kernel and Mesa patches pending

Great experience working with the Nouveau community

- Plans to keep contributing support for future Tegra SoCs

NVIDIA

Thank you!

https://github.com/NVIDIA/tegra-nouveau-rootfs

NVIDIA.