GRATE

LIBERATING NVIDIA'S TEGRA GPU

February 2013 - Erik "kusma" Faye-Lund

kusmabite@gmail.com / @kusmabite

WHO AM I

- About 20 years of graphics programming experience
  - 10 years professionally
  - Former driver-developer at Falanx / ARM's Mali team
  - Involved in the development of OpenGL ES 1.1 and 2.0
- Active open source contributor
  - Lots of Git patches
  - Linux, Android, Angle, Mesa, ...
- Demo scener

WHAT IS GRATE

An effort to reverse engineer the Tegra GPU
...and eventually to create open source drivers for it.
Probably the furthest behind of the ARM SoC reverse engineered driver efforts

DISCLAIMER

Everything in this presentation is based on reverse-engineering
Most information presented **might** be wrong

THE HISTORY SO FAR

- In the summer/fall of 2012, I got envyous of the Lima-guys, so I decided to start looking into Tegra
- Around FOSDEM 2013, I had:
  - Command list capturing and parsing
  - Envytools-style RNNDB descriptions of most OpenGL ES 2.0 non-shader state
  - A very rough fragment shader disassembler
  - Reverse enginered the rough interface to the shader-compiler
- Got bored with it

ENTER THIERRY

- A month later, Luc told me to get my ass on IRC
- Turns out, while I was procrastinating Thierry Reding had picked up the ball:
  - Linux DRI/KMS
  - LibDRM
  - Got command-stream replay working
  - Started on a DDX-driver
  - Even did the initial work on a Gallium driver!
- Then Thierry got hired by NVIDIA to maintain the DRI driver
- I'm slowly trying to follow in his footsteps
  - However, my biggest interest is reverse-engineering

AWESOME WORK, THIERRY!

CURRENT STATUS

TEGRA 2

- This is the core I've focused on
- Command stream dumping
- Basic rendering through command stream replay
  - Can modify a lot of state by tampering with the command stream
- Upstream Linux DRM driver
- Downstream libDRM support
- Very, very unfinished downstream Mesa/Gallium driver
  - Can only do glClear and glReadPixels with GR2D

TEGRA 3

Replay seems to just work. Identical 3D core?

TEGRA 4

- Some additional registers discovered
- Not **strictly** compatible?
  - But modified Tegra 2 command-streams have been replayed

TEGRA K1

- Kepler based
  - Only the 3D core, lacks most other components of GeForce
- No work done
- Maybe something for Nouveau instead?
- Won't be covered further in this talk

DEMO (?)

HARDWARE

TEGRA 2 GPU OVERVIEW

- Code named AR20
- Immediate-mode renderer
- Consists of (at least) three components:
  - GR2D
  - GR3D
  - Video
- Clients are programmed through Host1x
  - DMA engine for writing registers
- Proprietary OpenGL ES drivers

GR2D

- Documented in the publically available TRM
  - Requires signing up and agreeing to an EULA
- Example source code available
- Blits / fills / patterns
- Tiling / linear source and destination
- Stretching
- Rotation / flipping
  - 90° / 180° / 270°
- Blending
- CSAA resolve
- ROP3
- Lots more, see TRM

GR3D

- Non-unified shader
- Performs blending in the fragment shader
- 16 bit depth buffer
  - Tegra 4 also supports 24 bit depth
- 16 render targets (including depth/stencil)
- Occlusion queries
- Texturing:
  - Floating-point textures
  - Texture arrays
  - Anisotropic filtering
  - ETC1, S3TC, DXT1, LATC
  - Non-pow2-ish textures
- GL_OES_standard_derivatives
- GL_NV_draw_path

VIDEO

- No work so far
- I'm not a video-expert
- Up for grabs!

VERTEX SHADER ISA

- NV30 subset
  - 4 component vector ALU
  - scalar SFU
- No control flow
- Straight forward to generate code for
- Share code with Nouveau?

FRAGMENT SHADER ISA

- Registers are 1 x 20 bit floating-point or 2 x 10-bit fixed-point
- At least 3 separate instruction streams:
  - ALU - Arithmetic/Logic Unit
  - MFU - Multi-Function Unit
    - Varying interpolation
    - Complex function evaluation
    - Not executed in the same clock?
  - TEX - Texturing Unit
  - EXPORT?
  - Others? (import for spilling?)
- No control flow

FRAGMENT SHADER ISA: ALU

- Pretty much understood
- Instructions comes in packets
  - Can perform 4 scalar ops per instruction packet
  - Or 3 scalar ops with 2 x 20 bit / 4 x 10 bit embedded constants
- Glorified MAD
  - 1 destination, 3 source operands
  - D = A * B + C
  - D = A * B + C * C
  - D = (A + C) * B
  - D += ...
- MIN/MAX/CSEL
- Predicate instructions
- Saturate result
- Absolute / negate source operands
- Scale source operands by 2, result by 0.5, 2 or 4

FRAGMENT SHADER ISA: MFU

- Probably based on "A High-Performance Area-Efficient Multifuction Interpolator", Oberman et. al, 2005
- Complex function evaluation pretty much understood
  - NOP, RCP, RSQ, LG2, EX2, SQRT, SIN, COS, FRC
  - PREEX2, PRESIN, PRECOS
    - Not unlike NVIDIA with two-step trig
- Varying write is still a mystery :(
  - This is a major blocker
  - Help, please!

FRAGMENT SHADER ISA: TEX

- Somewhat understood, but...
  - Not clear where texture coordinates come from
  - Progressing on this feels pointless without varying writes understood
- Seems simple
  - 2D textures and cube maps lookups compile bitwise identical
  - No need to normalize cubemap inputs

FRAGMENT SHADER ISA: EXPORT

- Render-target index found
- The rest is pretty much a mystery :(

HELP WANTED!

TODO

- finish/upstream libDRM patches
- X.org DDX driver
  - GR2D is completely documented
  - Helps hardening the libDRM interface
- Reverse engineering
  - Varying writes!!!
  - Fragment shader exporting
  - Register spilling
  - ...
- Mesa / Gallium driver
  - Easier said than done ;)

QUESTIONS?