Lima driver status update

XDC 2019

Connor Abbott, Erico Nunes, Vasily Khoruzhick
Overview

- Lima is an open source graphics driver which supports Mali 400/450 embedded GPUs from ARM via reverse engineering
- Upstreamed in mesa 19.1 and linux kernel 5.2
Mali 400/450

- ARM claims it to be one of the world’s most shipped mobile GPUs
- OpenGL ES 2.0
- Tiling rendering model
- Two different cores, two instruction sets, two compilers
- Up to 8 PP cores, up to 2 GP cores (Mali 450)
- No integers
- Offline shader compiler available
- Many devices don’t have distribution support due to requiring blobs and out of tree kernel driver
- Mali GPU does not feature display controllers (HDMI, LCD, etc).
History

- yuq’s recent efforts prior to the merge to upstream (2017)
  [https://github.com/yuq/mesa-lima](https://github.com/yuq/mesa-lima) (now deprecated)
  [https://gitlab.freedesktop.org/lima](https://gitlab.freedesktop.org/lima)
- Lots of code copy-and-pasted from the initial project, especially on command stream assembling
- Reference documentation for compiler development, such as reverse-engineered instruction sets, and tools such as a disassembler, also come from the initial project
Lima driver

- gallium driver
- nir compiler
- renderonly/kmsro for display
- gpir and ppir
Vertex shader: gpir

- Insane scalar VLIW architecture
- Pipeline details are exposed in ISA
- Difficult to write a compiler for
Mali GP

- Designed before GLES2 was a thing (~2005)
- Design goal: minimum size/power usage
- Similar to early non-unified desktop GPU's
  - Fixed maximum # of instructions (512)
  - No texture fetches or direct memory access of any kind
- Processes one vertex at a time
- Yet insanely hard to write a compiler for!

*From the original Falanx GP product spec*
Register Bypassing

Fetch → Decode → Read → Execute → Write

add R0, R1, R2
add R1, R3, R4
add R2, R4, R5
Register Bypassing

Add $R0$, $R1$, $R2$

Add $R1$, $R3$, $R4$

Add $R2$, $R4$, $R5$
Explicit Bypassing

\[\text{add } R0, ^1, ^0 \quad \text{add } R1/^1, R3, R4 \quad \text{add } R2/^0, R4, R5\]
Mali GP Pipeline

- Six execution units: 2x multiply, 2x add, 1x lookup table, 1x misc.
- Lookup table used to compute reciprocal, exponent, log, invsqrt
  - Extremely weird, meant to be used as part of a fixed sequence
- Registers, everything else can only be read/written in 4-component vectors
- Read/write ports are decoupled from execution units in the ISA
- 1x register read port, 1x register/attribute read, 1x register/varying write
  - Can write xy and zw components to different registers
Mali GP Pipeline

Register File

Attributes

Uniforms/Scratch

Read mux

2× Add

2× Mul

LUT/Cplx

Misc/Pass

Write mux

Varyings

×2

×4

×4

×4

×2
Mali GP

- Straightforward to translate ARB_vertex_program assembly programs, even GLSL
  - Split vec4 instructions in half, schedule halves independently, resolve read/write conflicts using explicit bypass network
- But, you don't get good utilization of resources
- Not good enough for actual GLSL programs, due to 512 instruction limit!
Move Threading

\[
\text{add } r_1.x \text{ } r_2.x \text{ } r_2.y, \ldots \\
\ldots \\
\ldots \\
mul \text{ } r_3.x \text{ } r_1.x \text{ } r_3.x, \ldots \\
\ldots \\
\text{add } ^0 \text{ } r_2.x \text{ } r_2.y, \ldots \\
\ldots \\
\ldots \\
\text{mov } ^1 \text{ } ^0, \ldots \\
mul \text{ } r_3.x \text{ } ^1 \text{ } r_3.x, \ldots \\
\ldots \
\]
Mali GP

- From the offline shader compiler user guide:

"MaliGP2 has only limited internal bandwidth between its registers and execution units. In some rare cases, the register allocator for MaliGP2 in the shader compiler runs into a situation where there are more operations executed in one cycle than can be fed from the registers simultaneously. The compiler aborts the compilation in this case."  

¯\_(ツ)_/¯
Mali GP

- Most complex part by far is the scheduler (~1800 lines)
  - Guaranteed to never fail due to running out of register slots, unlike ARM's compiler
  - Produces shaders that are usually on-par with ARM

- Current gpir status (mesa 19.3)
  - All possible nir ops implemented
  - Control flow is implemented
  - No spilling or indirect variable access yet
Mali PP

- VLIW vec4 architecture with some ops being scalar-only
- FP16-only
  - Sampler coords can be 32-bit if loaded directly from varying
  - Any operations on sampler coords reduce precision to fp16
  - Some piglit tests fail due to insufficient precision.
- Only 6 vec4 registers
  - Probably more in silicon, since according to ARM there can be up to 128 threads
- Pipeline registers between some modules
  - Saves a real reg in some cases
Mali PP

Can do in one instruction:

- Load vec4 uniform or temporary
- Load vec4 varying
- 1 sampler
- Up to 14 flops
  - Vec4 addition
  - Vec4 multiplication
  - Scalar addition
  - Scalar multiplication
  - Vec4 - scalar multiplication or transcendental fns
- Store temporary or FB fetch
Mali PP Pipeline

Uniform or temp fetch

Const

Vec Mul

Vec Acc

Scl Mul

Scl Acc

Varying fetch

Sampler

Combine

Branch

R0
R1
R2
R3
R4
R5
Mali PP

- lower: nir instructions don't always map directly to PP instructions, add/remove some nodes when required.
- schedule: create pp instructions from the result of the lowering above.
- register allocation: try to do RA, if it does not succeed and we need to spill, insert standalone temporary-load/store instructions to the already-scheduled result.
- codegen: not many tricks there, just generate the binary.
Mali PP

- Way simpler than gpir
- Attempts to utilize pipeline registers whenever possible
  - Uniforms loads are duplicated for each consumer
    - According to ARM site there’s no penalty to fetch it from cache and 1 cycle penalty to fetch it from memory
  - Constants are duplicated for each consumer
  - Varying loads and samplers are duplicated for each basic block where it’s used
    - Reduces number of live values
- Regalloc is not optimal
  - Scheduler tries to put as many ops as possible in one instruction
  - But it’s not aware of register pressure
Mali PP

- Current ppir status (mesa 19.3)
  - Most of possible nir ops implemented
  - Control flow implemented
  - Spilling support
  - Can deal with quite complex shaders
Kernel lima driver

- Merged in lima 5.2
- GEM driver
- DRM scheduler
- PRIME to display
  - Works with sun4i, rockchip, meson, exynos
- Not too many changes since it was merged
Things that are supported now?

- **Works**
  - kmscube, kodi, mythtv, mpv

- **With restrictions**
  - Piglit, deqp, glmark2, q3a

- **With several restrictions**
  - X, sway, Weston, Desktop environments (KDE Plasma)
Going forward

- **Current state**
  - Developers working on piglit tests
  - Demo applications work
  - Many applications still unsupported

- **Some features are still missing for lima to become a complete graphics driver.**
  - X11 is slow since we always render whole surface (i.e. we don’t drop unmodified tiles when scissor test is enabled)

- **Android mostly works**
  - Some rendering artifacts
Going forward

- Project status is at
  - https://gitlab.freedesktop.org/lima/web
- We have a mailing list
  - https://lists.freedesktop.org/mailman/listinfo/lima
- IRC channel #lima on freenode is fairly active
Demo

Demo time!