SOLUTION TO SHADER RECOMPILES IN RADEONSI

SEPTEMBER 2015

PROBLEM

- Shaders are compiled in draw calls
  - Emulating certain features in shaders
- Drivers keep shaders in some intermediate representation
- And insert additional code based on the states
- While compiling, everything stops
- Number of state combinations is exponential

EMULATED STATES

AMD

- Fragment shader:
  - Conversion to colorbuffer formats (RGBA32, RGBA FP16, …)
  - Alpha-test
  - Selecting between front and back colors
  - gl_FragColor
  - GL_ALPHA_TO_ONE
  - Polygon stippling
  - Line & polygon smoothing
  - Point smoothing
  - Fragment color clamping

EMULATED STATES, CONT.

- Vertex shader:
  - Loading inputs from vertex buffers manually
  - Vertex color clamping

IDEA

AMD

- Observation:
  - All states can be applied at the beginning or end of shaders
  - At link time, compile application shaders
  - At draw time, append any shader bytecode needed
- 3 shader sections:
  - Prolog section
  - Main section (application shader)
  - Epilog section
- Concatenate them

FRAGMENT SHADER EPILOG

Color outputs are expected in r0, r1, …

```
out0 = r0;
out1 = r1;
```

FRAGMENT SHADER EPILOG

If we need alpha-test:

```
if (!alphafunc(r0.w, alpharef)) discard;

out0 = r0;
out1 = r1;
```

FRAGMENT SHADER EPILOG

If we need color clamping:

```
r0 = clamp(r0, 0, 1);
r1 = clamp(r1, 0, 1);
if (!alphafunc(r0.w, alpharef)) discard;

out0 = r0;
out1 = r1;
```

FRAGMENT SHADER EPILOG

If we need polygon stippling:

```
r0 = clamp(r0, 0, 1);
r1 = clamp(r1, 0, 1);
if (!alphafunc(r0.w, alpharef)) discard;
if (texture2D(stipple, gl_FragCoord.xy / 32).x < 0.5) discard;
out0 = r0;
out1 = r1;
```

FRAGMENT SHADER EPILOG

AMD

If we need smoothing:

```
r0 = clamp(r0, 0, 1);
r1 = clamp(r1, 0, 1);
if (!alphafunc(r0.w, alpharef)) discard;
if (texture2D(stipple, gl_FragCoord.xy / 32).x < 0.5) discard;
r0.w *= coverageMask; // popcount(gl_SampleMaskIn) / gl_NumSamples
r1.w *= coverageMask;
out0 = r0;
out1 = r1;
```

FRAGMENT SHADER EPILOG

AMD

If color conversion is required:

```
r0 = clamp(r0, 0, 1);
r1 = clamp(r1, 0, 1);
if (!alphafunc(r0.w, alpharef)) discard;
if (texture2D(stipple, gl_FragCoord.xy / 32).x < 0.5) discard;
r0.w *= coverageMask; // popcount(gl_SampleMaskIn) / gl_NumSamples
r1.w *= coverageMask;
r0.xy = vec2(packHalf2x16(r0.xy), packHalf2x16(r0.zw));
r1.xy = vec2(packHalf2x16(r1.xy), packHalf2x16(r1.zw));
out0 = r0;
out1 = r1;
```

FRAGMENT SHADER EPILOG

If GL_ALPHA_TO_ONE is enabled:

```
r0 = clamp(r0, 0, 1);
r1 = clamp(r1, 0, 1);
if (!alphafunc(r0.w, alpharef)) discard;
if (texture2D(stipple, gl_FragCoord.xy / 32).x < 0.5) discard;
r0.w *= coverageMask; // popcount(gl_SampleMaskIn) / gl_NumSamples
r1.w *= coverageMask;
r0.w = 1;
r0.xy = vec2(packHalf2x16(r0.xy), packHalf2x16(r0.zw));
r1.xy = vec2(packHalf2x16(r1.xy), packHalf2x16(r1.zw));
out0 = r0;
out1 = r1;
```

FRAGMENT SHADER PROLOG

AMD

- Only contains two-side color selection
- Decreases performance if done always

- 3 scenarios:
  - Two-side colors are enabled:
    - Select colors based on gl_FrontFacing
    - Store them into registers r0, r1
  - Two-side colors are disabled:
    - Just copy front colors into r0, r1
  - No color inputs => prolog is empty
- Application shader should read colors from r0, r1

COMPILING PROLOGS/EPILOGS

- Still have to be compiled in draw calls
  - Can be slow

- Use an assembler instead of the compiler
  - Our LLVM backend has an assembler too

VERTEX SHADER INPUTS

AMD

- R600 had fetch shader
- Removed since GCN

- Current implementation:
  - One buffer per input
  - Instance divisor == 0: Fetch BaseVertex + VertexID
  - Instance divisor != 0: Fetch StartInstance + (InstanceID / instance divisor)

VERTEX SHADER PROLOG

AMD

- Emulate fetch shader with prolog section
  - Drawback: can’t move loads to hide latencies, register usage
- Instead, only calculate load addresses:
  - Prolog writes the addresses to r0,r1, …
  - Main shader section executes the loads

VERTEX SHADER EPILOG?

AMD

- Radeon has 3 ways to write VS outputs:
  - For rasterizer
  - For geometry shader
  - For tessellation control shader

- Don’t use an epilog
- OpenGL sometimes knows which shader follows
- If not, compile all 3 variants with 3 threads in parallel
- Piglit only: Compile on demand in draw calls

- Vertex color clamping: use conditional assignment

MESA STATE TRACKER

AMD

- Middle-end, translates shaders from GLSL IR into TGSI
- Does that in draw calls
- State dependencies for draw calls:
  - Center vs sample interpolation
    - Instead, select coordinates with conditional assignment
  - Vertex and fragment color clamping
  - GL rendering context

- Any dependencies should be dealt with in drivers
- Other drivers will benefit too
  - GLSL->TGSI always done at link time

IF GAMES COMPILE TOO LATE

AMD

- Compiling at link time doesn’t help
- Use shader cache
- 1 shader variant => shader cache in core Mesa

- If games compile early => don’t need it

SKIP MESA OPTIMIZATIONS?

AMD

- Our LLVM backend can do most optimizations
  - No need to do them in Mesa
- Mesa/GLSL passes we do need:
  - Demoting inputs/outputs to local variables (dead code elimination?)
  - Function inlining
  - Breaking built-in input/output arrays into variables

Questions?

THANK YOU

DISCLAIMER & ATTRIBUTION


AMD