What's new in ir3

Job Noorman

XDC'25

What *is* ir3?

- Mesa compiler backend for Qualcomm Adreno GPUs
- Used by Freedreno (Gallium) and Turnip (Vulkan)
- Supports Adreno a3xx - a7xx
- **I**ntermediate **R**epresentation **3** (because a3xx)

Adreno ISA

- SIMT, 64/128 threads per wave
- Scalar ISA
- Load store ISA
- Native 16/32-bit FP/int ALUs/SFUs
- Arbitrary branching and predication

Adreno ISA: execution units

- Vector ALU (simple ops)
    - Fixed latency, managed by compiler (nop insertion)
- Special function unit (`sqrt`, `rcp`,…)
    - Short variable latency, managed by compiler (`(ss)` sync flags)
- Scalar ALU
    - Uniform ops (mostly preamble)
    - No latency, sync when reading from vector ALU
    - Same opcodes as Vector ALU, selected based on register class
- Texture processor, load/store unit
    - Long variable latency, managed by compiler (`(sy)` sync flags)

Adreno ISA: register classes

- 48×4 32-bit "full" (`r0.x`,...) and 16-bit "half" (`hr0.x,...`) GPRs
  - GPR space shared among waves, limits parallelism
  - a6xx+: overlap with bottom half of "full" registers
- 8×4 32-bit "shared" uniform GPRs, (`r48.x`,...)
- 1-bit predicate registers (`p0.[xyzw]`) for branching
- Constant registers (`c0.x`,...)
  - Uniform memory, addressable like registers by many instructions
  - Lowered UBOs, push constants, driver params,...

Compilation flow

1. Generate and optimize NIR
2. Instruction selection (SSA)
    - Create CFG
3. Optimizations
    - copy propagation, conversion folding, DCE,...
4. Pre-RA scheduling (register pressure)
5. RA (out-of-SSA)
6. Post-RA scheduling (latency)
7. Legalization
8. Assembly (isaspec)

What is *new* in ir3

- Status: most ISA features up to a7xx have been implemented
- New a6xx features
  - Scalar ALU and early preamble
  - Predicate registers and predication
  - Full (e.g., clustered) and improved (i.e., native) subgroup ops
  - 64-bit integers
  - Repeated instructions (simulated vector ops)
- New a7xx features
  - 64-bit atomics
  - 8-bit storage/integers
  - Ray tracing
  - Aliased tex srcs
  - Aliased render targets

Repeated instructions

Simulated vector instructions

```
32x4 %3 = iadd %1, %2.xxxx
```

```
(rpt3)add.u r3.x, (r)r1.x, r2.x
```

```
add.u r3.x, r1.x, r2.x
add.u r3.y, r1.y, r2.x
add.u r3.z, r1.z, r2.x
add.u r3.w, r1.w, r2.x
```

Repeated instructions: implementation

1. Ingest vectorized NIR (`nir_opt_vectorize`)
2. Emit vector ops as scalar ops linked together in "repeat groups"
3. For now: completely ignore in optimizations/scheduling
   - Assumption: code size less important than other optimizations
4. Pre-RA: create merge sets for repeat groups
   - Try to allocate consecutive regs but don't force it
5. Post-RA: merge instructions in repeat groups if assigned regs allow it

Aliased tex srcs: problem

- Texture ops need large number of consecutive GPRs
- May lead to fragmentation of the register file
- Increases register pressure even for constants/immediates
- Fragmentation may lead to moves

```
r0 = textureLod(vec2(x, 0), lod)
```

```
; x in r1.z, lod in r1.x
; saml first src must have {x, y, lod} in consecutive regs
mov r1.w, 0
mov r2.x, r1.w
; make sure regs are written
(rpt1)nop
saml (xyzw)r0.x, r1.z
```

Register aliases: solution

- a7xx introduced "alias registers" to remap registers

```
; x in r1.z, lod in r1.x
; saml first src must have {x, y, lod} in consecutive regs
alias.tex r1.w, 0
alias.tex r2.x, r1.w
saml (xyzw)r0.x, r1.z
```

- Advantages over moves:
  - Aliases do not occupy GPR space (MaxWaves+, MOVs-)
  - Do not need synchronization (NOPs-)
  - Can reference constants/immediates (MOVs-, GPRs-)

ir3 tooling: compiler flags

`IR3_SHADER_DEBUG` (environment variable)

- `disasm`, `optmsgs` (pass results), `ramsgs` (RA), `schedmsgs` (scheduling),…
- Sync issues
  - `fullsync` (`(ss)(sy)nop` everywhere)
  - `fullnop` (`(rpt5)nop` everywhere)
- Disabling features: `noaliastex`, `noearlypreamble`,…

ir3 tooling: shader overrides

`IR3_SHADER_OVERRIDE_PATH` (environment variable)

- After shader compilation:
  - If `$IR3_SHADER_OVERRIDE_PATH/$shader_id.asm` exists
  - Assemble and replace shader binary
- Allows fast iteration for codegen bug hunting/fixing
- Custom assembler:
  - Hand coded flex/bison
  - Full ir3 support
  - Helpful macros (e.g., `@fullsync{start,end}`)

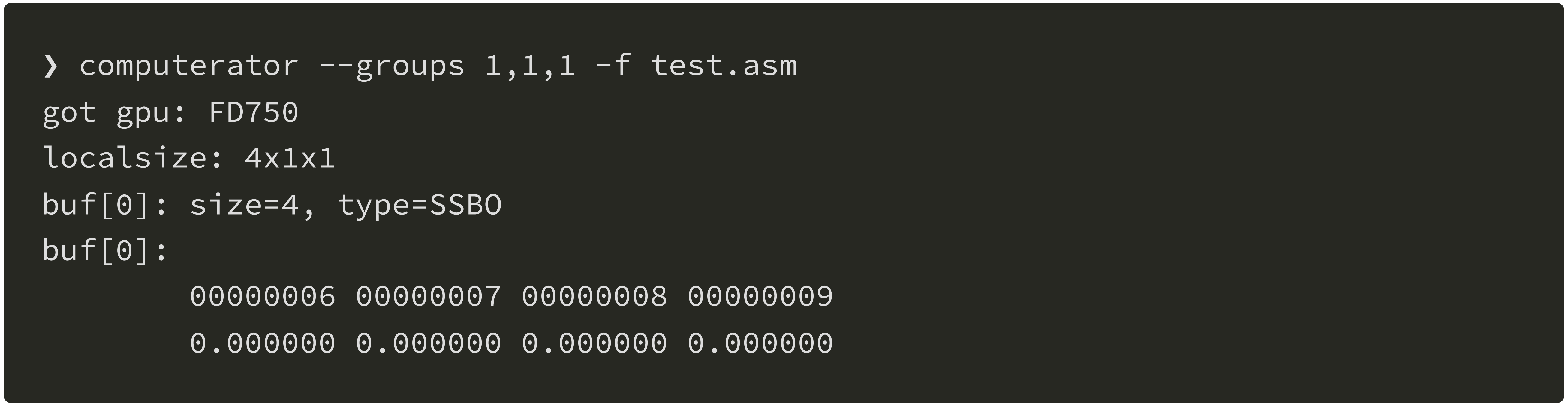
ir3 tooling: computerator

- Use assembler to generate compute shader
- Set up minimal GPU state to dispatch
- *Extremely* useful for reverse engineering

```
@localsize 4, 1, 1
@buf 4 1, 2, 3, 4
@invocationid(r0.x)

ldib.b.untyped.1d.u32.4.imm r1.x, r0.x, 0
(sy)add.u r1.x, r1.x, 5
(rpt2)nop
stib.b.untyped.1d.u32.4.imm r1.x, r0.x, 0
end
```

ir3 tooling: computerator



```
❯ computerator --groups 1,1,1 -f test.asm
got gpu: FD750
localsize: 4x1x1
buf[0]: size=4, type=SSBO
buf[0]:
        00000006 00000007 00000008 00000009
        0.000000 0.000000 0.000000 0.000000
```

Questions?

igalia