Turnip: Improving performance without compromising correctness

Danylo Piliaiev

2025-10-01

Quick update on Turnip

- Adreno 750 is the main optimization target
- With a few outstanding MRs, Turnip supports almost everything HW is intended to support
- Performance is on par with the prop driver on many d3d11 titles
- Correctness is better than prop driver

Challenges

- Hardware supports several rendering modes:
  - Direct (sysmem) rendering
  - Tiled (gmem) rendering
  - Direct/Tiled + Concurrent Binning
- A fair number of issues are seen only in one mode
  - Ideally, tests should run at least once for each mode
- Fun fact: CTS conformance submission doesn't care about that, so you can have entirely broken rendering in one mode and be "conformant"

Challenges

- **Not much testing of the driver in the wild.**

- CTS doesn't test well a number of features we have:
  - Low-Resolution-Z, Concurrent Binning, Concurrent Resolves
- CTS doesn't test big FBs which is important for Tiled rendering

Testing for correctness

- Single-frame traces are the most practical way to test

Single-frame Trace PROS	Multi-frame Trace PROS
Easier to gather	Closer to real workload
Much faster to replay	Can be a better proxy for performance
Easy to check render stability	Reproduce multi-frame issues

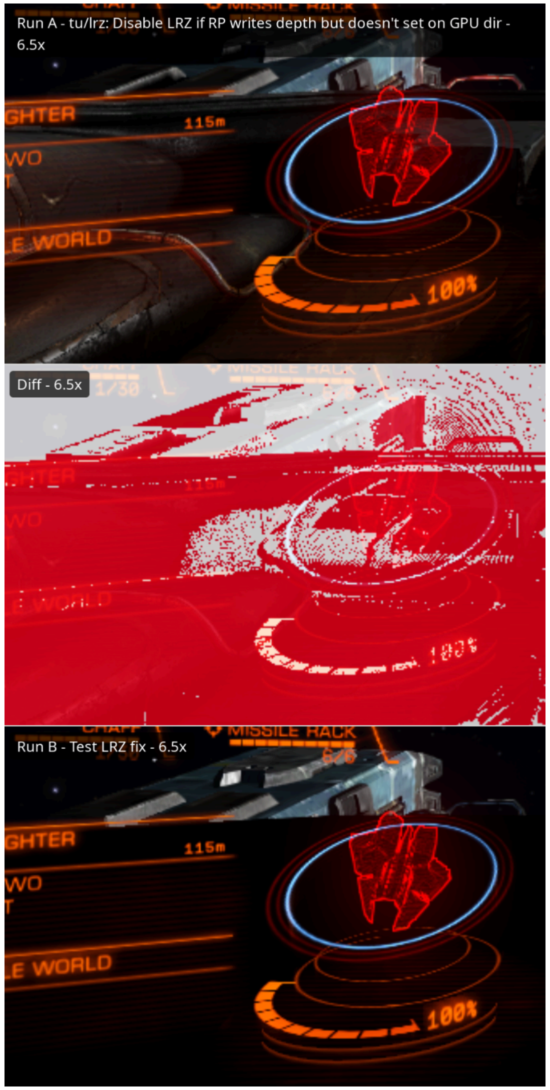
Testing for correctness

Single-frame Trace CONS	Multi-frame Trace CONS
Less likely to repro multi-frame issues	Takes much more space
Fewer things are tested per trace	Takes much more time to execute
Narrower performance testing	Can take more time to gather

Turnip testing

What do we test

- Frame stability: replaying a frame in a loop yields the same result
    - Great at finding undefined behavior
- Compare render results between runs
- Catch GPU faults
- Gather and compare performance data



Run A - tu/lrz: Disable LRZ if RP writes depth but doesn't set on GPU dir - 6.5x

Diff - 6.5x

Run B - Test LRZ fix - 6.5x

What do we run?

- D3D11, D3D9, D3D8, GL, VK single-frame traces
- A default CI run tests ~360 traces
- Takes about 2h 30m per run on a single HDK
- Runs two nightly passes: forced direct and forced tiling

Our findings

- It's not rare for an issue to happen only on a **single** trace
- Looping the frame is effective in catching issues
- Visual inspection of diffs, without golden frames, is good enough
  - Well, if you don't have many GPUs to test...

Tooling limitations

- Cutting-edge features may not be supported by tracing tools
  - Solution: We don't support them =)
- Traces compatibility between GPUs:
  - OpenGL and D3D8-11 are mostly already compatible
  - Vulkan needs VK profiles with intersection of capabilities
  - D3D12 ???
- Not every api has a tool for single-frame captures
  - We added trimming support to apitrace for D3D9 and D3D8

Perf regression testing

- Single-frame traces are also good enough to track performance!
  - RenderDoc captures won't correspond 1:1 to in-game perf
  - The effect on perf may differ in-game
- The direction of perf change is the same
- The magnitude can be lower, but that's not important

How?

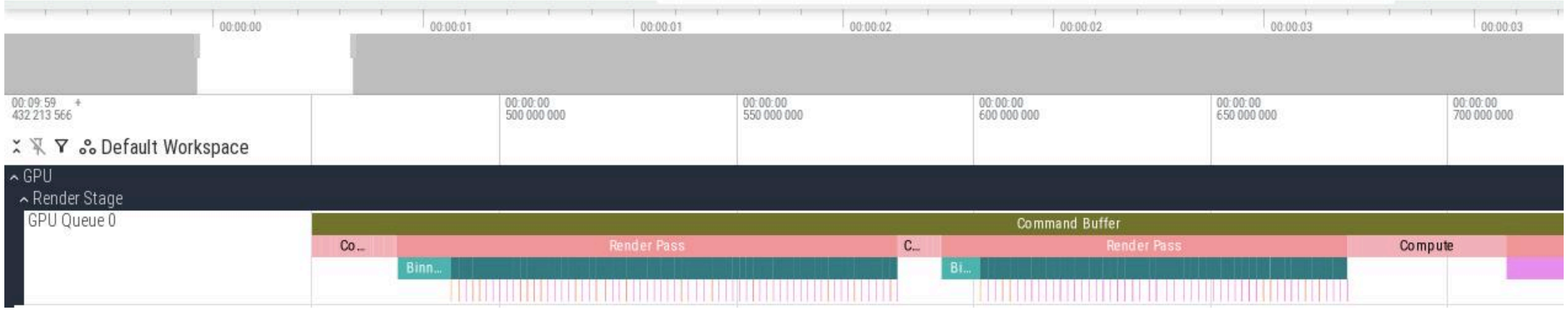
- Looping a single frame doesn't saturate the GPU
- RenderDoc takes a lot of time copying resources on the GPU
- Our solution is to use `u_trace` to measure renderpasses and dispatches
- Loop the single frame 10 times; discard the first and last
  - Renderpass and dispatch durations are averaged
- Standalone tool is `gpu-trace-perf` by Emma Anholt

u_trace

- Already integrated in:
    - Freedreno, Turnip, RadeonSI, ANV, Iris, PanVK, RADV (WIP)
- Easy to integrate, if you haven't - just do it!
- The only requirement - ability to write timestamps from CP

u_trace benefits

- Better `gpu-trace-perf` support
- Relatively easy integration with `perfetto`

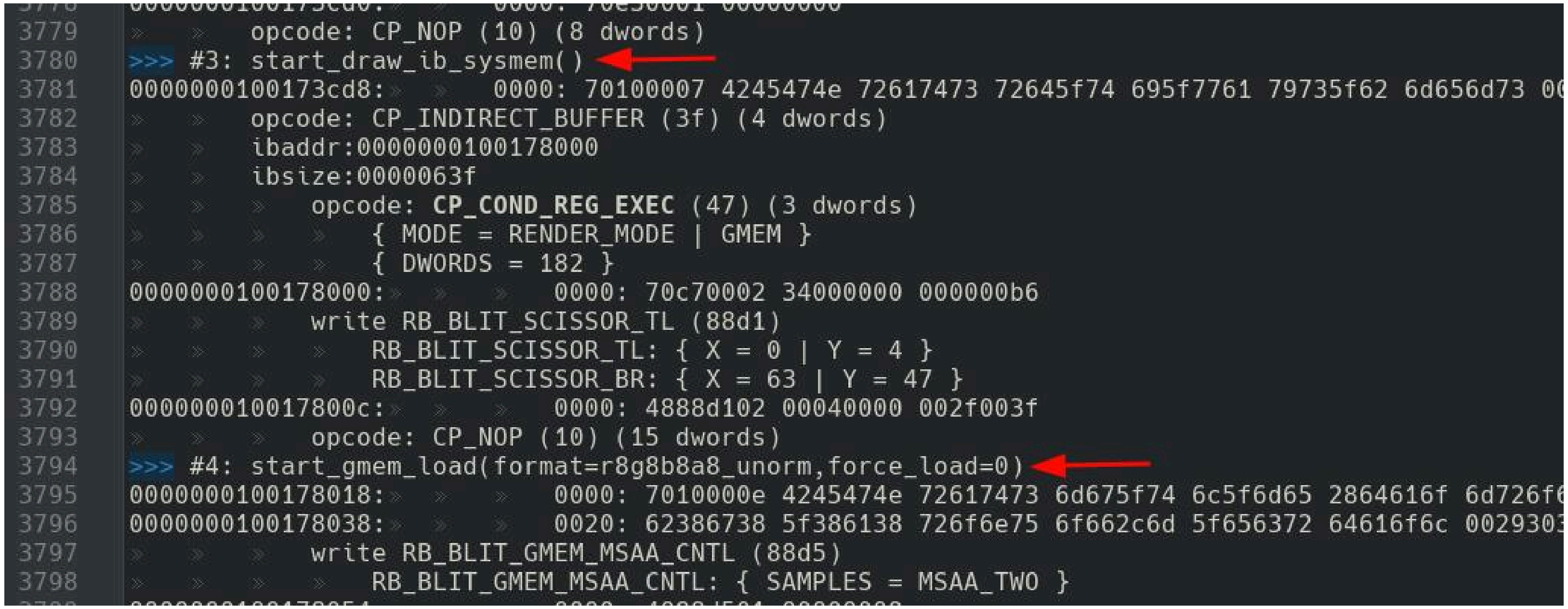


- Dump performance data into CSV or JSON formats

```
frame,batch,time_ns,event,
0,83,199544728760,start_compute,indirect=0, unaligned=0,
0,83,199544732296,end_compute,
0,83,199544732452,start_render_pass,maxSamples=1, clearCP
0,83,199546358284,end_render_pass,tiledRender=false, tili
```

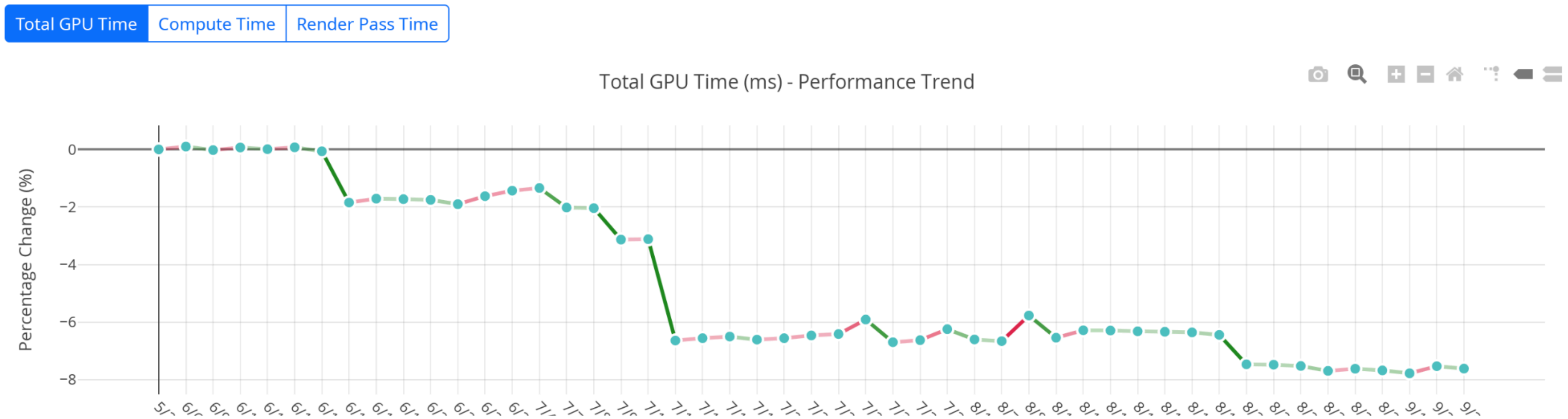
u_trace benefits

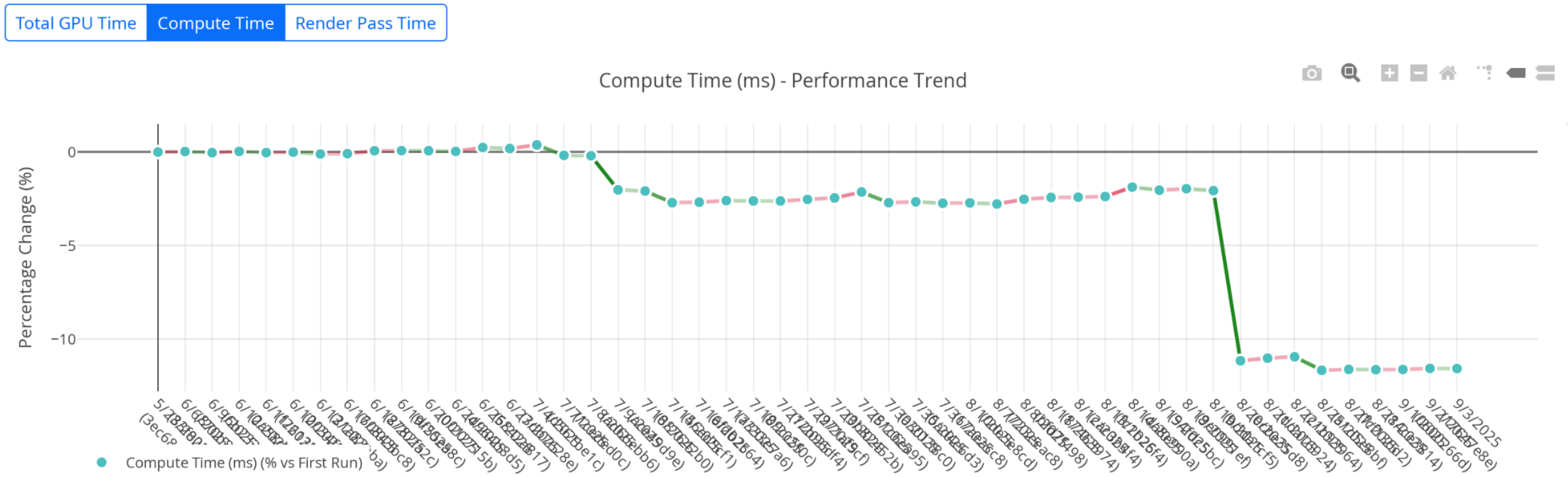
- Displaying values of indirect params, or other relevant memory
- Optional markers in command stream dump helping to navigate it:



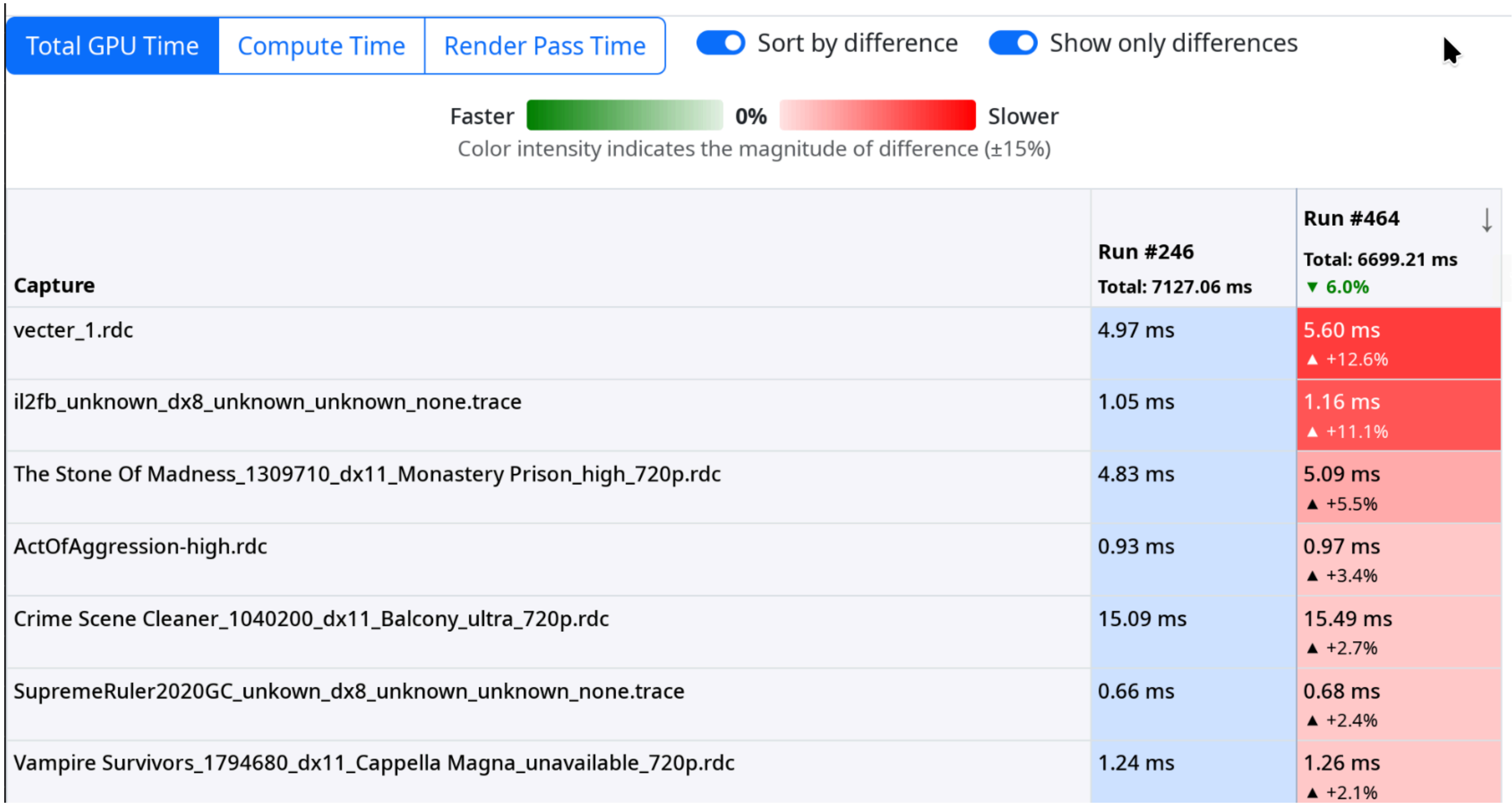
```
0000000100173cd0:        0000: 70e50001 00000000
                opcode: CP_NOP (10) (8 dwords)
>>> #3: start_draw_ib_sysmem()
0000000100173cd8:        0000: 70100007 4245474e 72617473 72645f74 695f7761 79735f62 6d656d73 00
                opcode: CP_INDIRECT_BUFFER (3f) (4 dwords)
                ibaddr:0000000100178000
                ibsize:0000063f
                        opcode: CP_COND_REG_EXEC (47) (3 dwords)
                                { MODE = RENDER_MODE | GMEM }
                                { DWORDS = 182 }
0000000100178000:                0000: 70c70002 34000000 000000b6
                        write RB_BLIT_SCISSOR_TL (88d1)
                                RB_BLIT_SCISSOR_TL: { X = 0 | Y = 4 }
                                RB_BLIT_SCISSOR_BR: { X = 63 | Y = 47 }
000000010017800c:                0000: 4888d102 00040000 002f003f
                        opcode: CP_NOP (10) (15 dwords)
>>> #4: start_gmem_load(format=r8g8b8a8_unorm,force_load=0)
0000000100178018:                0000: 7010000e 4245474e 72617473 6d675f74 6c5f6d65 2864616f 6d726f6
0000000100178038:                0020: 62386738 5f386138 726f6e75 6f662c6d 5f656372 64616f6c 0029303
                        write RB_BLIT_GMEM_MSAA_CNTL (88d5)
                                RB_BLIT_GMEM_MSAA_CNTL: { SAMPLES = MSAA_TWO }
```

Performance tracking





Per-trace perf difference



● Sort by difference ● Show only differences

Faster 🟩 0% 🟥 Slower

Color intensity indicates the magnitude of difference (±15%)

Capture	Run #246 Total: 7127.06 ms	Run #464 ↓ Total: 6699.21 ms ▼ 6.0%
vecter_1.rdc	4.97 ms	5.60 ms ▲ +12.6%
il2fb_unknown_dx8_unknown_unknown_none.trace	1.05 ms	1.16 ms ▲ +11.1%
The Stone Of Madness_1309710_dx11_Monastery Prison_high_720p.rdc	4.83 ms	5.09 ms ▲ +5.5%
ActOfAggression-high.rdc	0.93 ms	0.97 ms ▲ +3.4%
Crime Scene Cleaner_1040200_dx11_Balcony_ultra_720p.rdc	15.09 ms	15.49 ms ▲ +2.7%
SupremeRuler2020GC_unkown_dx8_unknown_unknown_none.trace	0.66 ms	0.68 ms ▲ +2.4%
Vampire Survivors_1794680_dx11_Cappella Magna_unavailable_720p.rdc	1.24 ms	1.26 ms ▲ +2.1%

Measurements limitations

- We don't track non-{RP/dispatch} performance
- We don't account for translation layer performance changes
- Real games often use lots of RAM bandwidth, which affects performance
- Individual trace perf can be rather noisy (depends on the trace)

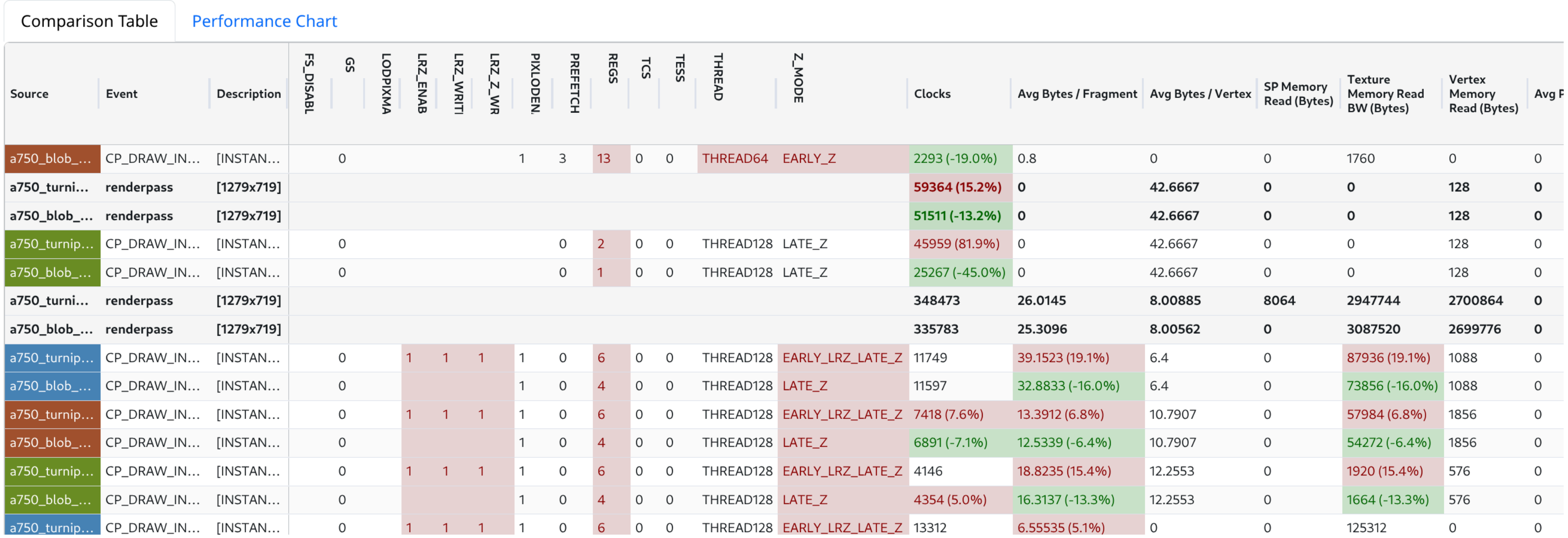
Improving performance

Improving performance

- Adreno is a mobile GPU and has perf pain points in different places compared to desktop GPUs
- We were clearly behind the proprietary driver in performance
- Knowing that we are slower in specific games doesn't help much:
    - No way to compare perf per-draw
    - Hard to compare even per renderpass

Improving performance

- We've built tooling to compare Turnip against the proprietary driver down to the draw-call level:



Comparison Table | Performance Chart

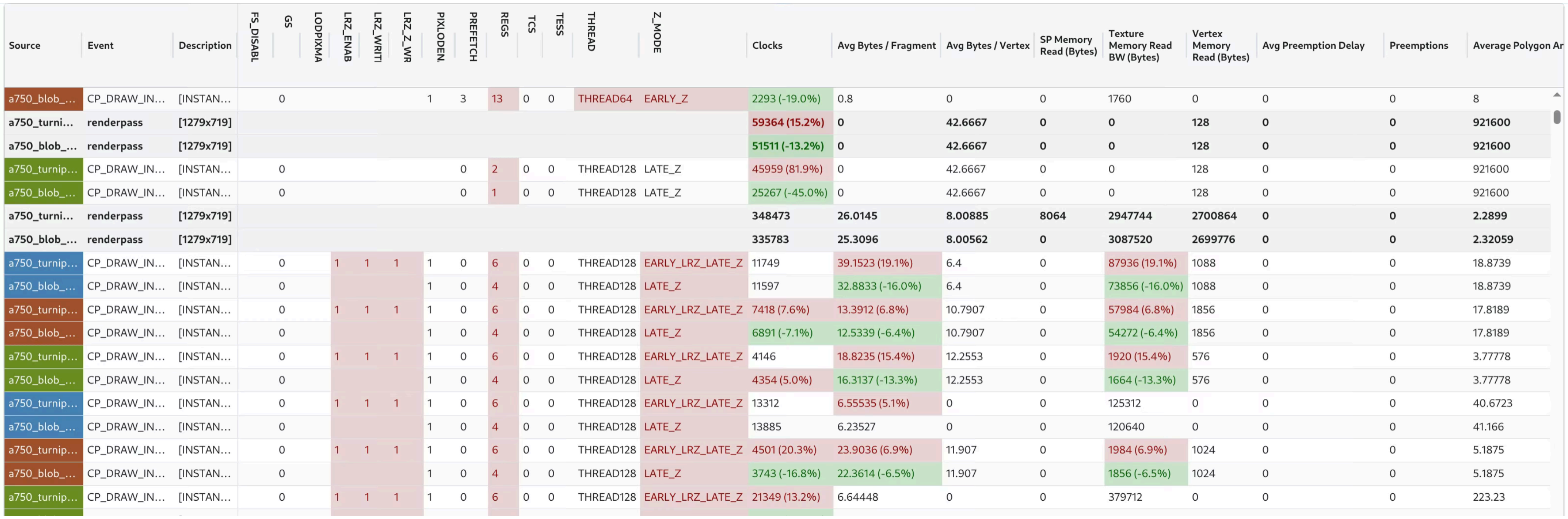
Source	Event	Description	FS_DISABL	GS	LODPIXMA	LRZ_ENAB	LRZ_WRITI	LRZ_Z_WR	PIXLODEN	PREFETCH	REGS	TCS	TESS	THREAD	Z_MODE	Clocks	Avg Bytes / Fragment	Avg Bytes / Vertex	SP Memory Read (Bytes)	Texture Memory Read BW (Bytes)	Vertex Memory Read (Bytes)	Avg F
a750_blob_…	CP_DRAW_IN…	[INSTAN…		0					1	3	13	0	0	THREAD64	EARLY_Z	2293 (-19.0%)	0.8	0	0	1760	0	0
a750_turni…	renderpass	[1279x719]														59364 (15.2%)	0	42.6667	0	0	128	0
a750_blob_…	renderpass	[1279x719]														51511 (-13.2%)	0	42.6667	0	0	128	0
a750_turnip…	CP_DRAW_IN…	[INSTAN…		0						0	2	0	0	THREAD128	LATE_Z	45959 (81.9%)	0	42.6667	0	0	128	0
a750_blob_…	CP_DRAW_IN…	[INSTAN…		0						0	1	0	0	THREAD128	LATE_Z	25267 (-45.0%)	0	42.6667	0	0	128	0
a750_turni…	renderpass	[1279x719]														348473	26.0145	8.00885	8064	2947744	2700864	0
a750_blob_…	renderpass	[1279x719]														335783	25.3096	8.00562	0	3087520	2699776	0
a750_turnip…	CP_DRAW_IN…	[INSTAN…		0		1	1	1	1	0	6	0	0	THREAD128	EARLY_LRZ_LATE_Z	11749	39.1523 (19.1%)	6.4	0	87936 (19.1%)	1088	0
a750_blob_…	CP_DRAW_IN…	[INSTAN…		0					1	0	4	0	0	THREAD128	LATE_Z	11597	32.8833 (-16.0%)	6.4	0	73856 (-16.0%)	1088	0
a750_turnip…	CP_DRAW_IN…	[INSTAN…		0		1	1	1	1	0	6	0	0	THREAD128	EARLY_LRZ_LATE_Z	7418 (7.6%)	13.3912 (6.8%)	10.7907	0	57984 (6.8%)	1856	0
a750_blob_…	CP_DRAW_IN…	[INSTAN…		0					1	0	4	0	0	THREAD128	LATE_Z	6891 (-7.1%)	12.5339 (-6.4%)	10.7907	0	54272 (-6.4%)	1856	0
a750_turnip…	CP_DRAW_IN…	[INSTAN…		0		1	1	1	1	0	6	0	0	THREAD128	EARLY_LRZ_LATE_Z	4146	18.8235 (15.4%)	12.2553	0	1920 (15.4%)	576	0
a750_blob_…	CP_DRAW_IN…	[INSTAN…		0					1	0	4	0	0	THREAD128	LATE_Z	4354 (5.0%)	16.3137 (-13.3%)	12.2553	0	1664 (-13.3%)	576	0
a750_turnip…	CP_DRAW_IN…	[INSTAN…		0		1	1	1	1	0	6	0	0	THREAD128	EARLY_LRZ_LATE_Z	13312	6.55535 (5.1%)	0	0	125312	0	0

igalia

How it was done

- Create compatible with both drivers `.gfxr` traces
- Intercept cmdbuf submissions to the kernel
- Add instrumentation around cmdbufs, RPs, draws/dispatches
- Dump registers important for perf
- Gather perf counters
- Merge all perf counters together into a huge spreadsheet

How many perf counters?



Source	Event	Description	FS_DISABL	GS	LODPIXMA	LRZ_ENAB	LRZ_WRITI	LRZ_Z_WR	PIXLODEN	PREFETCH	REGS	TCS	TESS	THREAD	Z_MODE	Clocks	Avg Bytes / Fragment	Avg Bytes / Vertex	SP Memory Read (Bytes)	Texture Memory Read BW (Bytes)	Vertex Memory Read (Bytes)	Avg Preemption Delay	Preemptions	Average Polygon Ar
a750_blob_...	CP_DRAW_IN...	[INSTAN...		0					1	3	13	0	0	THREAD64	EARLY_Z	2293 (-19.0%)	0.8	0	0	1760	0	0	0	8
a750_turni...	**renderpass**	**[1279x719]**														**59364 (15.2%)**	**0**	**42.6667**	**0**	**0**	**128**	**0**	**0**	**921600**
a750_blob_...	**renderpass**	**[1279x719]**														**51511 (-13.2%)**	**0**	**42.6667**	**0**	**0**	**128**	**0**	**0**	**921600**
a750_turnip...	CP_DRAW_IN...	[INSTAN...		0						0	2	0	0	THREAD128	LATE_Z	45959 (81.9%)	0	42.6667	0	0	128	0	0	921600
a750_blob_...	CP_DRAW_IN...	[INSTAN...		0						0	1	0	0	THREAD128	LATE_Z	25267 (-45.0%)	0	42.6667	0	0	128	0	0	921600
a750_turni...	**renderpass**	**[1279x719]**														**348473**	**26.0145**	**8.00885**	**8064**	**2947744**	**2700864**	**0**	**0**	**2.2899**
a750_blob_...	**renderpass**	**[1279x719]**														**335783**	**25.3096**	**8.00562**	**0**	**3087520**	**2699776**	**0**	**0**	**2.32059**
a750_turnip...	CP_DRAW_IN...	[INSTAN...		0		1	1	1	1	0	6	0	0	THREAD128	EARLY_LRZ_LATE_Z	11749	39.1523 (19.1%)	6.4	0	87936 (19.1%)	1088	0	0	18.8739
a750_blob_...	CP_DRAW_IN...	[INSTAN...		0					1	0	4	0	0	THREAD128	LATE_Z	11597	32.8833 (-16.0%)	6.4	0	73856 (-16.0%)	1088	0	0	18.8739
a750_turnip...	CP_DRAW_IN...	[INSTAN...		0		1	1	1	1	0	6	0	0	THREAD128	EARLY_LRZ_LATE_Z	7418 (7.6%)	13.3912 (6.8%)	10.7907	0	57984 (6.8%)	1856	0	0	17.8189
a750_blob_...	CP_DRAW_IN...	[INSTAN...		0					1	0	4	0	0	THREAD128	LATE_Z	6891 (-7.1%)	12.5339 (-6.4%)	10.7907	0	54272 (-6.4%)	1856	0	0	17.8189
a750_turnip...	CP_DRAW_IN...	[INSTAN...		0		1	1	1	1	0	6	0	0	THREAD128	EARLY_LRZ_LATE_Z	4146	18.8235 (15.4%)	12.2553	0	1920 (15.4%)	576	0	0	3.77778
a750_blob_...	CP_DRAW_IN...	[INSTAN...		0					1	0	4	0	0	THREAD128	LATE_Z	4354 (5.0%)	16.3137 (-13.3%)	12.2553	0	1664 (-13.3%)	576	0	0	3.77778
a750_turnip...	CP_DRAW_IN...	[INSTAN...		0		1	1	1	1	0	6	0	0	THREAD128	EARLY_LRZ_LATE_Z	13312	6.55535 (5.1%)	0	0	125312	0	0	0	40.6723
a750_blob_...	CP_DRAW_IN...	[INSTAN...		0					1	0	4	0	0	THREAD128	LATE_Z	13885	6.23527	0	0	120640	0	0	0	41.166
a750_turnip...	CP_DRAW_IN...	[INSTAN...		0		1	1	1	1	0	6	0	0	THREAD128	EARLY_LRZ_LATE_Z	4501 (20.3%)	23.9036 (6.9%)	11.907	0	1984 (6.9%)	1024	0	0	5.1875
a750_blob_...	CP_DRAW_IN...	[INSTAN...		0					1	0	4	0	0	THREAD128	LATE_Z	3743 (-16.8%)	22.3614 (-6.5%)	11.907	0	1856 (-6.5%)	1024	0	0	5.1875
a750_turnip...	CP_DRAW_IN...	[INSTAN...		0		1	1	1	1	0	6	0	0	THREAD128	EARLY_LRZ_LATE_Z	21349 (13.2%)	6.64448	0	0	379712	0	0	0	223.23

Is it even useful?

- Displaying critical registers next to the draw call helped a lot!
  - We fixed several issues with Low-Resolution-Z (LRZ)
  - Enabled fast path for depth-only draw calls
  - Found out when to fall back to a smaller FS wave size

Is it even useful?

- Perf counters were much more of a mixed bag:
  - Helped identify a few issues
  - There are a lot of them, measuring who knows what
  - A lot of counters are interdependent in some way
  - It's just hard to compare across totally different drivers

Future Plans CI/Perf

- Add masking of inherently unstable frames in CI
- Measure perf counters in CI to see subtle effects of optimizations
- Test D3D12 traces in CI
- Do more micro-benchmark comparisons with proprietary driver

Q&A

We're hiring!

igalia.com/jobs/

igalia