etnaviv

The wonderful world of performance counters

whoami

Christian Gmeiner

Long term etnaviv developer

agenda

GPU performance counter

Vivante GPUs

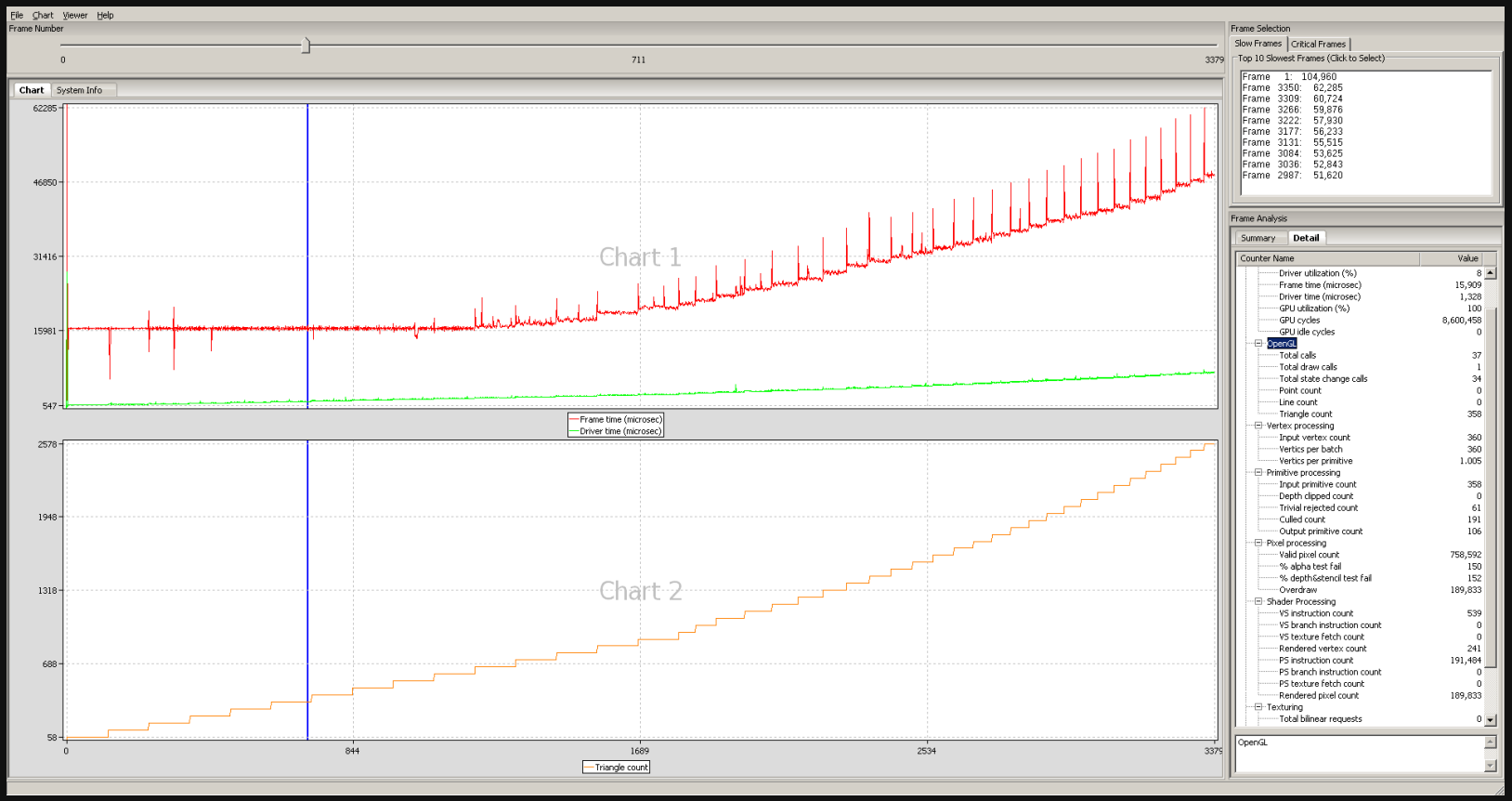
binary blob

etnaviv

Future

GPU performance counter

Can help to analyze the performance and execution characteristics of applications.



Vivante GPUs

Limitations

No access via command stream to the performance counters
exception: occlusion queries

There might be hardware support in coming GPUs

gcvFEATURE_PROBE

https://source.codeaurora.org/external/imx/linux-imx

branch: imx_5.4.3_2.0.0

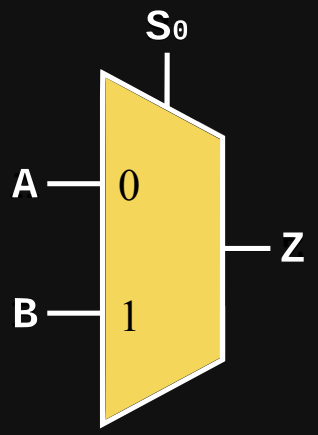
```
typedef enum _gceProbeStatus
{
    gcvPROBE_Disabled = 0,
    gcvPROBE_Paused = 1,
    gcvPROBE_Enabled = 2,
}
gceProbeStatus;

typedef enum _gceProbeCmd
{
    gcvPROBECMD_BEGIN = 0,
    gcvPROBECMD_PAUSE = 1,
    gcvPROBECMD_RESUME = 2,
    gcvPROBECMD_END = 3,
}
gceProbeCmd;

typedef struct _gcsPROBESTATES
{
    gceProbeStatus              status;
    gctUINT32                   probeAddress;
}gcsPROBESTATES;
```

How to access performance counter

simple register read

mux config write + register read

per pixel pipe counter values



take care of clock gating and debug register access

galcore interface

gcvHAL_GET_PROFILE_SETTING

gcvHAL_SET_PROFILE_SETTING

gcvHAL_READ_PROFILER_REGISTER_SETTING

gcvHAL_READ_ALL_PROFILE_REGISTERS_PART1

gcvHAL_READ_ALL_PROFILE_REGISTERS_PART2

gcvHAL_PROFILE_REGISTERS_2D

https://source.codeaurora.org/external/imx/linux-imx branch: imx_5.4.3_2.0.0

galcore as register documentation

```
/* HW profile information. */
typedef struct _gcsPROFILER_COUNTERS_PART1
{
    gctUINT32       gpuTotalRead64BytesPerFrame;
    gctUINT32       gpuTotalWrite64BytesPerFrame;

    /* SH */
    ..
    gctUINT32       ps_branch_inst_counter;

    ..
}
gcsPROFILER_COUNTERS_PART1;
```

```
gcmkONERROR(gckOS_WriteRegisterEx(Hardware->os, Hardware->core, 0x00470,
((((gctUINT32) (0)) & ~(((gctUINT32) (((gctUINT32) ((((1 ?
 31:24) - (0 ?
 31:24) + 1) == 32) ?
 ~0U : (~(~0U << ((1 ?
 31:24) - (0 ?
 31:24) + 1))))))) << (0 ?
 31:24))) | (((gctUINT32) ((gctUINT32) (13) & ((gctUINT32) ((((1 ?
 31:24) - (0 ?
 31:24) + 1) == 32) ?
 ~0U : (~(~0U << ((1 ? 31:24) - (0 ? 31:24) + 1))))))) << (0 ? 31:24))) ));
gcmkONERROR(gckOS_ReadRegisterEx(Hardware->os, Hardware->core, 0x0045C,
 &profiler_part1->ps_branch_inst_counter));
```

etnaviv

How to support performance counter

Go the galcore way and provide an ioctl to readout all counter in one go

- \+ quite easy to implement - see galcore
- − different ioctl's per pipe (2D and 3D)
- − ABI compatibility if new counters are supported
- − no relation to a specific draw call

How to support performance counter

Emulate the missing counter access via command stream

- takes more time to get API/ABI right
- \+ one interface for all pipes
- \+ ABI compatibility if new counters are supported
- \+ relation to a specific draw call
- \+ we can support one-shot-readouts

Emulation via command stream

Extend the in-kernel command buffer to

- trigger an event aka IRQ
- stop the FE
- process the IRQ on the driver and some work
- restart the FE

This called a **syncpoint** in the etnaviv kernel driver.

kernel interface

There is a need to extend the current kernel interface to

- get a list of possible counters
- configure counter readout via submit ioctl
- expose counter's data to userspace

query domains and signals

```
static const struct etnaviv_pm_domain doms_3d[] = {
        {
                .name = "SH",
                .profile_read = VIVS_MC_PROFILE_SH_READ,
                .profile_config = VIVS_MC_PROFILE_CONFIG0,
                .nr_signals = 9,
                .signal = (const struct etnaviv_pm_signal[]) {
                        {
                                "SHADER_CYCLES",
                                VIVS_MC_PROFILE_CONFIG0_SH_SHADER_CYCLES,
                                &perf_reg_read
                        },
                        ...
                }
        },
    ...
};
```

query domains and signals

```
1  struct drm_etnaviv_pm_domain {
2          __u32 pipe;         /* in */
3          __u8  iter;         /* in/out, select pm domain at index iter */
4          __u8  id;           /* out, id of domain */
5          __u16 nr_signals; /* out, how many signals does this domain provide */
6          char  name[64];   /* out, name of domain */
7  };
8
9  struct drm_etnaviv_pm_signal {
10         __u32 pipe;         /* in */
11         __u8  domain;       /* in, pm domain index */
12         __u8  pad;
13         __u16 iter;         /* in/out, select pm source at index iter */
14         __u16 id;           /* out, id of signal */
15         char  name[64];   /* out, name of domain */
16 };
17
18 #define DRM_ETNAVIV_PM_QUERY_DOM        0x0a
19 #define DRM_ETNAVIV_PM_QUERY_SIG        0x0b
```

extending submit ioctl

```
1  /* performance monitor request (pmr) */
2  #define ETNA_PM_PROCESS_PRE             0x0001
3  #define ETNA_PM_PROCESS_POST            0x0002
4  struct drm_etnaviv_gem_submit_pmr {
5          __u32 flags;          /* in, when to process request (ETNA_PM_PROCESS_x) */
6          __u8  domain;         /* in, pm domain */
7          __u8  pad;
8          __u16 signal;         /* in, pm signal */
9          __u32 sequence;       /* in, sequence number */
10         __u32 read_offset;    /* in, offset from read_bo */
11         __u32 read_idx;       /* in, index of read_bo buffer */
12 };
```

Make it possible to store a counter value at a defined offset of a provided bo.

Also provide a way to define when the sampling should happen.

extending submit ioctl

```
1 struct drm_etnaviv_gem_submit {
2         __u32 fence;          /* out */
3         __u32 pipe;           /* in */
4         ...
5         __u64 pmrs;           /* in, ptr to array of submit_pmr's */
6         __u32 nr_pmrs;        /* in, number of submit_pmr's */
7         __u32 pad;
8 };
```

Make it possible to provide multiple performance monitor requests (pmr).

Integration in gallium

each pipe_query has own bo

->begin_query(..): add a pmr to the submit with PM_PROCESS_PRE

->end_query(..): add a pmr to the submit with PM_PROCESS_POST

->get_query_result(..): cpu_prep(..), check sequence number, cpu_fini(..)

->flush(..) we do a end-flush-begin dance

Integration in gallium

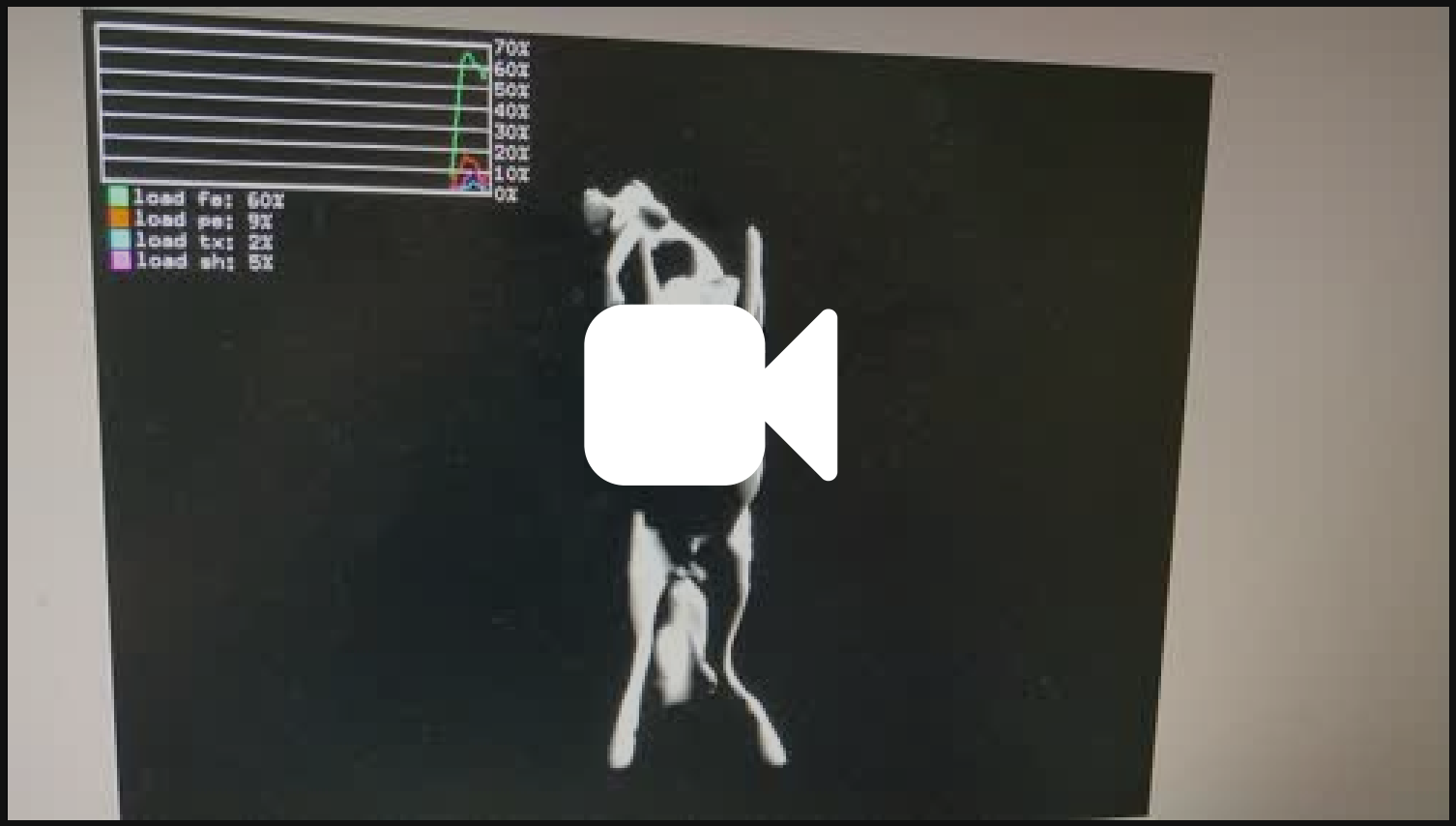
GALLIUM_HUD

AMD_performance_monitor

Future goals

GPU utilization

Mostly work in progress



Bandwidth values

- FE_READ_BANDWIDTH
- MMU_READ_BANDWIDTH
- BLT_READ_BANDWIDTH
- SH0_READ_BANDWIDTH
- SH1_READ_BANDWIDTH
- PE_WRITE_BANDWIDTH
- BLT_WRITE_BANDWIDTH
- SH0_WRITE_BANDWIDTH
- SH1_WRITE_BANDWIDTH

perfetto

System profiling, app tracing and trace analysis

work in progress patches

Thank you!