Status update of NVIDIA's performance counters for Nouveau

Samuel Pitoiset

Nouveau & X.Org developer enthusiast

September 17th, 2015

Who am I ?

Open source enthusiasm

- Nouveau & mesa contributor
  - performance counters (most of the time) & small GL bug fixes
- Google Summer of Code student in 2013 & 2014
- XDC talk last year in Bordeaux, France

Real life job

- Got my master degree last year
- HPC engineer at INRIA, Bordeaux
  - developing a source-to-source OpenMP compiler (Clang)

Summary

1. Introduction
   - What are performance counters ?
   - NVIDIA's perf counters
   - NVIDIA's profiling tools
2. Case study
3. Reverse engineering
4. Nouveau & mesa
5. APIs & Tools
6. Conclusion

What are performance counters ?

Performance counters

- are blocks in modern processors that monitor their activity
- count low-level hardware events such as cache hits/misses

Why use them ?

- to analyze the bottlenecks of 3D and GPGPU applications
- to dynamically adjust the performance level of the GPU

How to use them ?

- GUIs like NVIDIA Nsight and Linux Graphics Debugger
- APIs like NVIDIA CUPTI and PerfKit
- OpenGL extensions like GL_AMD_performance_monitor

NVIDIA's performance counters

Two groups of counters exposed

- **compute counters** for GPGPU applications
  - ex: warps_launched, divergent_branch ...
- **graphics counters** for 3D applications
  - ex: shader_busy, texture_busy ...

Different types of counters

- **global** counters
  - collect activities regardless of the context
- **local** counters
  - collect activities per-context only

NVIDIA's profiling tools

Visual Profiler

- cross-platform performance profiling tools for CUDA apps
- based on CUPTI API (expose compute-related counters)

Nsight

- Visual Studio plugin for profiling GL/D3D apps (Windows)
- based on PerfKit API (expose graphics-related counters)

Linux Graphics Debugger

- performance profiling tools for GL apps (SIGGRAPH'15)
- expose graphics-related counters on Linux (yeah!)
  - unfortunately, no API like PerfKit is provided

Summary

1 Introduction

2 Case study
- Improve a GL app with NVIDIA's tools
- What about Nouveau ?

3 Reverse engineering

4 Nouveau & mesa

5 APIs & Tools

6 Conclusion

Improve a GL app

How to improve performance of a GL app using perf counters ?

Let's try **NVIDIA Linux Graphics Debugger**!

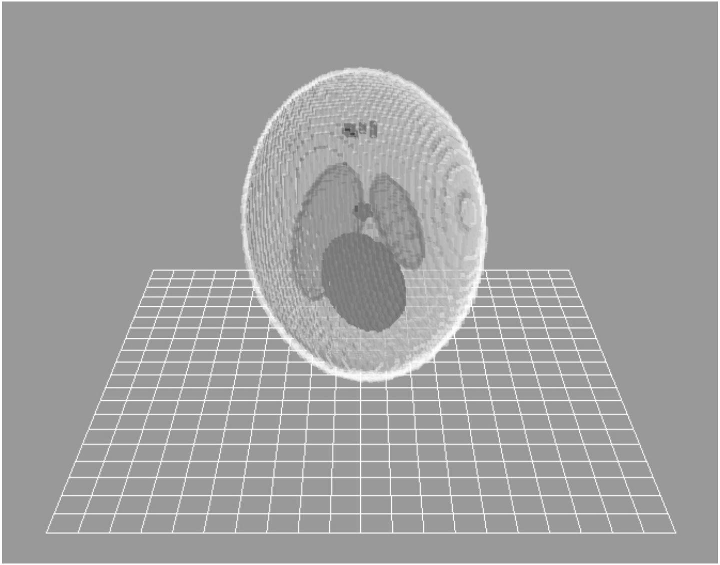


Figure : A brain rendered in OpenGL with 165786 voxels

Improve a GL app

Perf counters	Values
FPS	🟥 56

Improve a GL app

Perf counters	Values
FPS	56
geom_busy	1%

Improve a GL app

Perf counters	Values
FPS	56
geom_busy	1%
shader_busy	0.2%

Improve a GL app

Perf counters	Values
FPS	56
geom_busy	1%
shader_busy	0.2%
texture_busy	0.5%

Improve a GL app

Perf counters	Values
FPS	56
geom_busy	1%
shader_busy	0.2%
texture_busy	0.5%
ia_requests	350000

Improve a GL app

Perf counters	Values
FPS	56
geom_busy	1%
shader_busy	0.2%
texture_busy	0.5%
ia_requests	350000
l2_read_sysmem_sectors	200000

Improve a GL app

Perf counters	Values
FPS	56
geom_busy	1%
shader_busy	0.2%
texture_busy	0.5%
ia_requests	350000
l2_read_sysmem_sectors	200000

mmh...
l2_read_sysmem_sectors seems to **very high** and this is probably one of the bottlenecks!

Improve a GL app

Problem

- too many memory reads from the system memory
  - due to the GPU fetching the vertices at every frame

[1] There are probably other bottlenecks but this is just a basic example

Improve a GL app

Problem

- too many memory reads from the system memory
    - due to the GPU fetching the vertices at every frame

Solution

- use a vbo to store the vertices on the GPU

[1] There are probably other bottlenecks but this is just a basic example

Improve a GL app

Problem

- too many memory reads from the system memory
  - due to the GPU fetching the vertices at every frame

Solution

- use a vbo to store the vertices on the GPU

Perf counters	Without VBO	With VBO
FPS	56	470[1]
geom_busy	1%	1%
shader_busy	0.2%	0.2%
texture_busy	0.5%	0.5%
ia_requests	350000	250000
l2_read_sysmem_sectors	200000	35

[1] There are probably other bottlenecks but this is just a basic example

What about Nouveau ?

No tools like Linux Graphics Debugger!

... but things are going to change!

What about Nouveau ?

No tools like Linux Graphics Debugger!

... but things are going to change!

Perf counters project

- started since GSoC'13
- not a trivial project and a ton of work
  - reverse engineering (long and hard process)
  - kernel and userspace support (including APIs & tools)

What about Nouveau ?

No tools like Linux Graphics Debugger!

... but things are going to change!

Perf counters project

- started since GSoC'13
- not a trivial project and a ton of work
  - reverse engineering (long and hard process)
  - kernel and userspace support (including APIs & tools)

Goals & Benefits

- expose perf counters in a useful and decent manner
- help developers to find bottlenecks in their 3D applications.

Summary

1 Introduction

2 Case study

3 Reverse engineering
- Compute-related counters
- Graphics-related counters
- Current status

4 Nouveau & mesa

5 APIs & Tools

6 Conclusion

Compute-related counters

Requirements

- CUDA and CUPTI API (CUDA Profiling Tools Interface)
- valgrind-mmt and demmt (envytools)
- cupti_trace from envytools repository
    - tool which helped me a lot in the REing process

Compute-related counters

Requirements

- CUDA and CUPTI API (CUDA Profiling Tools Interface)
- valgrind-mmt and demmt (envytools)
- cupti_trace from envytools repository
  - tool which helped me a lot in the REing process

How does it work?

1. launch cupti_trace (ie. cupti_trace -a NVXX)
   - will automatically trace each hardware event exposed
2. grab a cup of coffee :) and wait few minutes
3. traces are now saved to your disk
4. analyze and document them

Graphics-related counters

Reverse engineering PerfKit on Windows

- really painful and very long process! :(
- no MMIO traces and no valgrind-mmt
- need to do it by hand (dump registers, etc)
  - very hard to find multiplexers

Graphics-related counters

Reverse engineering PerfKit on Windows

- really painful and very long process! :(
- no MMIO traces and no valgrind-mmt
- need to do it by hand (dump registers, etc)
  - very hard to find multiplexers

Reverse engineering LGD on Linux

- this Linux Graphics Debugger saved my brain! :)
- almost same process as compute-related counters;
  - but not automatically because it's a GUI.
- really easy to find multiplexers this time.

Current status

- **DONE** means it's fully reversed and documented
- **MOSTLY** means that some perf counters are reversed
- **WIP** means that I started the reverse engineering process
- **TODO** means that it's on my (long) todolist

Perf counters	Tesla	Fermi	Kepler	Maxwell
Graphics	MOSTLY[1]	DONE	WIP[2]	TODO
Compute	DONE	DONE	DONE	MOSTLY[3]

[1] Except per-context counters (requires PerfKit).

[2] Need to RE new counting modes.

[3] Only on GM107 and need to RE per-context counters logic.

Summary

1. Introduction
2. Case study
3. Reverse engineering
4. Nouveau & mesa
   - Kernel interface
   - Synchronization
5. APIs & Tools
6. Conclusion

Kernel interface

Why is a kernel interface needed ?

- because **global counters** have to be programmed via MMIO
  - only root or the kernel can write to them

What the interface has to do ?

- set up the configuration of counters
- poll counters
- expose counter's data to the userspace (readout)

Synchronization

Synchronizing operations

- CPU: ioctls
- GPU: **software methods**

Software method

- command added to the command stream of the GPU context
- upon reaching the command, the GPU is paused
- the CPU gets an IRQ and handles the command

Nouveau

Perfmon work

- expose low-level configuration of perf counters
  - include lot of signals/sources for Tesla, Fermi and Kepler
- allow to schedule/monitor perf counters from the userspace
  - based on nvif (ioctls interface)
- no Perf support is planned for now!

mesa

NV50 driver

- patches series already submitted to mesa-dev (pending)
  - because this requires a libdrm release with nvif support
- will expose around 30 global perf counters
- will enable GL_AMD_performance_monitor

NVC0 driver

- patches still in my local tree but almost ready
- will expose around 80 global perf counters for Fermi/Kepler

Summary

1. Introduction
2. Case study
3. Reverse engineering
4. Nouveau & mesa
5. APIs & Tools
    - GL_AMD_performance_monitor
    - Nouveau PerfKit
    - Apitrace
6. Conclusion

GL_AMD_performance_monitor

OpenGL extension

- based on pipe_query interface
- drivers need to expose a group of GPU counters to enable it

GL_AMD_performance_monitor

OpenGL extension

- based on pipe_query interface
- drivers need to expose a group of GPU counters to enable it

Current status

- **released in mesa 10.6**
- expose per-context counters on Fermi/Kepler
  - this requires compute support to launch kernels
- used by Apitrace for profiling frames (GSoC'15)

GL_AMD_performance_monitor

OpenGL extension

- based on pipe_query interface
- drivers need to expose a group of GPU counters to enable it

Current status

- **released in mesa 10.6**
- expose per-context counters on Fermi/Kepler
    - this requires compute support to launch kernels
- used by Apitrace for profiling frames (GSoC'15)

Cons

- do not support round robin sampling and multi-pass events
- do not fit well with NVIDIA hardware (obviously)

Nouveau PerfKit

Linux version of NVIDIA PerfKit

- built on top of mesa (as a Gallium state tracker like VDPAU)
- needed to reverse engineer the API (return codes, etc)
    - around 100 unit/functional test have been written
- implemented libperfkit with both Windows and Linux support

Nouveau PerfKit

Linux version of NVIDIA PerfKit

- built on top of mesa (as a Gallium state tracker like VDPAU)
- needed to reverse engineer the API (return codes, etc)
  - around 100 unit/functional test have been written
- implemented libperfkit with both Windows and Linux support

Pros

- allow support of round robin sampling and multi-pass events

Nouveau PerfKit

Linux version of NVIDIA PerfKit

- built on top of mesa (as a Gallium state tracker like VDPAU)
- needed to reverse engineer the API (return codes, etc)
    - around 100 unit/functional test have been written
- implemented libperfkit with both Windows and Linux support

Pros

- allow support of round robin sampling and multi-pass events

Current status

- RFC submitted in June (around 1700 LOC, still in review)
- will expose more perf counters than gl_amd_perfmon
- no users for now but Apitrace could use PerfKit

Apitrace

GSoC'15 project

- add support for performance counters in the profiling view
- project by Alex Tru (mentored by Martin Peres)

Apitrace

GSoC'15 project

- add support for performance counters in the profiling view
- project by Alex Tru (mentored by Martin Peres)

DONE (but still not upstream)

- abstraction system for profiling in glretrace
  - support for GL_AMD_perfmon and Intel_perfquery
  - allow to query and to monitor metrics

Apitrace

GSoC'15 project

- add support for performance counters in the profiling view
- project by Alex Tru (mentored by Martin Peres)

DONE (but still not upstream)

- abstraction system for profiling in glretrace
  - support for GL_AMD_perfmon and Intel_perfquery
  - allow to query and to monitor metrics

WIP

- profiling view improvements for qapitrace
  - some minor parts are done but very basic visualization

Apitrace

Let's go back to the case study but now with...

... **Apitrace and Nouveau!**

Apitrace/Nouveau

How to list available metrics?

- glretrace –list-metrics <trace>

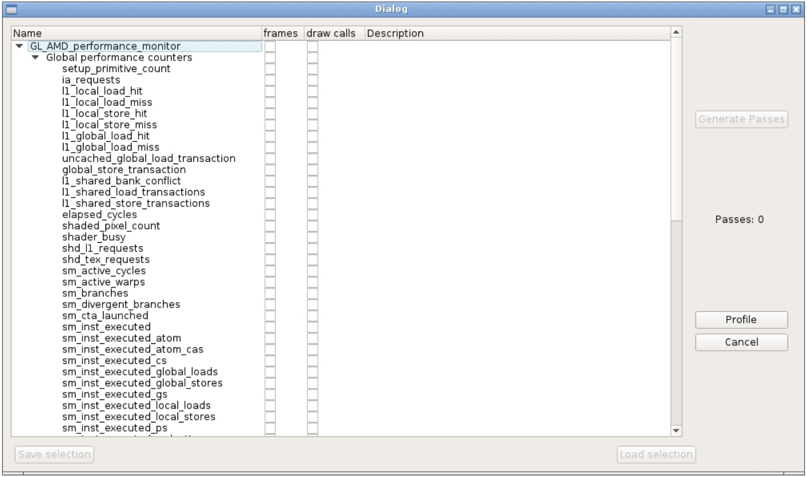
```
Backend GL_AMD_performance_monitor:

 Group #0: Global performance counters.
  Metric #0: shader_busy (type: CNT_TYPE_GENERIC, type: CNT_NUM_UINT64)
  Metric #1: ia_requests (type: CNT_TYPE_GENERIC, type: CNT_NUM_UINT64).
  Metric #2: texture_busy (type: CNT_TYPE_GENERIC, type: CNT_NUM_UINT64).

 Group #1: MP counters.
  Metric #0: active_cycles (type: CNT_TYPE_GENERIC, type: CNT_NUM_UINT64).
  Metric #1: active_warps (type: CNT_TYPE_GENERIC, type: CNT_NUM_UINT64).

Backend opengl:

 Group #0: CPU.
  Metric #0: CPU Start (type: CNT_TYPE_TIMESTAMP, type: CNT_NUM_INT64).
  Metric #1: CPU Duration (type: CNT_TYPE_DURATION, type: CNT_NUM_INT64).
```



Name	frames	draw calls	Description
GL_AMD_performance_monitor	□	□	
Global performance counters	□	□	
setup_primitive_count	□	□	
ia_requests	□	□	
l1_local_load_hit	□	□	
l1_local_load_miss	□	□	
l1_local_store_hit	□	□	
l1_local_store_miss	□	□	
l1_global_load_hit	□	□	
l1_global_load_miss	□	□	
uncached_global_load_transaction	□	□	
global_store_transaction	□	□	
l1_shared_bank_conflict	□	□	
l1_shared_load_transactions	□	□	
l1_shared_store_transactions	□	□	
elapsed_cycles	□	□	
shaded_pixel_count	□	□	
shader_busy	□	□	
shd_l1_requests	□	□	
shd_tex_requests	□	□	
sm_active_cycles	□	□	
sm_active_warps	□	□	
sm_branches	□	□	
sm_divergent_branches	□	□	
sm_cta_launched	□	□	
sm_inst_executed	□	□	
sm_inst_executed_atom	□	□	
sm_inst_executed_atom_cas	□	□	
sm_inst_executed_cs	□	□	
sm_inst_executed_global_loads	□	□	
sm_inst_executed_global_stores	□	□	
sm_inst_executed_gs	□	□	
sm_inst_executed_local_loads	□	□	
sm_inst_executed_local_stores	□	□	
sm_inst_executed_ps	□	□	

Passes: 0

Figure : List of available metrics in Apitrace

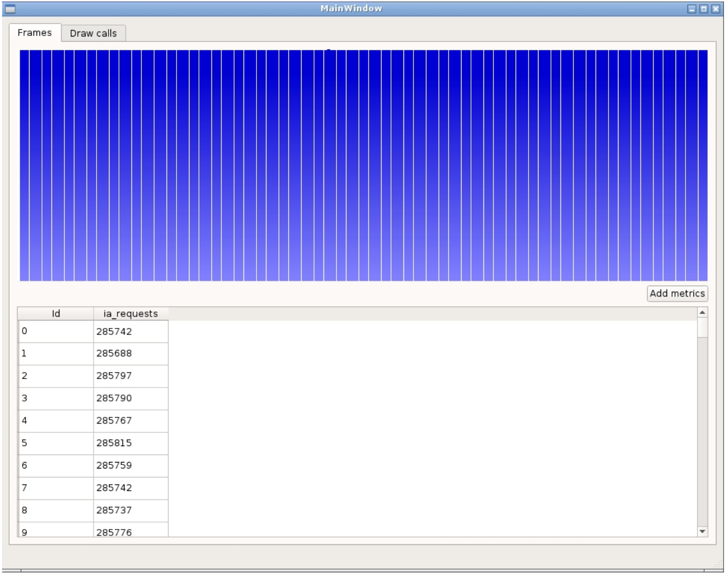
Apitrace/Nouveau

How to profile a GL app?

- glretrace –pframes="GL_AMD_perfmon: [0,65]" <trace>

```
#         ia_requests
frame     285734
frame     285799
frame     285793
frame     285763
frame     285762
frame     285809
frame     285800
frame     285744
frame     285743
frame     285796
frame     285893
frame     285818
frame     285754
frame     285804
frame     285762
frame     285763
frame     285813
frame     285804
frame     285815
frame     285747
frame     285754

Rendered 20 frames in 0.3365 secs, average of 59.4344 fps
```



Id	ia_requests
0	285742
1	285688
2	285797
3	285790
4	285767
5	285815
6	285759
7	285742
8	285737
9	285776

Figure : Very basic visualization with histograms in Apitrace

Apitrace/Nouveau

Perf counters	Without VBO	With VBO
geom_busy	7%	17%
shader_busy	0.5%	1%
texture_busy	2%	4%
ia_requests	371000	286000
l2_read_sysmem_sectors	193000	35
FPS	25[1]	160[1]

[1] Without reclocking

Summary

1. Introduction
2. Case study
3. Reverse engineering
4. Nouveau & mesa
5. APIs & Tools
6. Conclusion
   - Current status
   - Future work

Current status

Reverse engineering

- almost all perf counters on Tesla, Fermi and Kepler reversed

Current status

Reverse engineering

- almost all perf counters on Tesla, Fermi and Kepler reversed

Nouveau DRM & mesa

- perfmon work merged in Linux 4.3
- GL_AMD_performance_monitor merged in mesa 10.6

Current status

Reverse engineering

- almost all perf counters on Tesla, Fermi and Kepler reversed

Nouveau DRM & mesa

- perfmon work merged in Linux 4.3
- GL_AMD_performance_monitor merged in mesa 10.6

Userspace tools

- GL_AMD_perfmon used by Apitrace!
- perf counters are going to be exposed in a useful manner. :)

Future work

Short-term period

- add more signals & sources for Fermi and Kepler
- rework the software methods interface
- release libdrm with nvif support (Ben Skeggs)
- complete the support of perf counters in mesa
  - this will expose GL_amd_perfmon on Tesla
  - this will expose lot of perf counters on Tesla, Fermi and Kepler

Future work

Short-term period

- add more signals & sources for Fermi and Kepler
- rework the software methods interface
- release libdrm with nvif support (Ben Skeggs)
- complete the support of perf counters in mesa
    - this will expose GL_amd_perfmon on Tesla
    - this will expose lot of perf counters on Tesla, Fermi and Kepler

Long-term period

- finish implementation of Nouveau PerfKit
    - and make something use it (Apitrace?)
- reverse engineer Maxwell performance counters

Thanks!

I would like to thank the X.Org board members for my travel sponsorship!

Feel free to ask questions...