Pitfalls of benchmarking graphics applications for performance tracking

And how to address them?

Martin Peres

Intel Open Source Technology Center Finland

September 18, 2015

Summary

Introduction

Current situation

- Complex games/benchmarks are becoming available on Linux;
- Drivers are getting more complex as performance improves;
- Users now rely on Open Source drivers for performance.

Risks when merging new code

- Break previous functionalities / rendering;
- Break the performance of a game inadvertly;
- Improve the performance of one game but slow down others.

Introduction

Review does not catch everything - Real-life example

```
@@ -340,6 +340,10 @@ is_color_fast_clear_compatible(struct brw_context *brw,
                                const union gl_color_union *color)
 {
    if (_mesa_is_format_integer_color(format))
+      if (brw->gen >= 8) {
+         perf_debug("Integer fast clear not enabled for (%s)",
+                    _mesa_get_format_name(format));
+      }
       return false;
```

Result

- Up to 10% regression in some benchmarks;
- Took 13 days for the fix to reach upstream.

Introduction - Need for benchmarking

It is impossible to predict performance

Some factors affecting the performance:

- Data-and-code alignment and cache hierarchy/size;
- CPU and GPU schedulers;
- Samplers configuration;
- Hardware generation;
- Power budgets.

$\Rightarrow$ Need to benchmark all the platforms and games of interest.

Summary

Benchmarking

Different needs for benchmarking

- Developers: Run multiple experiments and compare them;
- QA:
  - Test patch series before they hit mainline;
  - Follow performance trends on mainline;
  - Create performance retrospective.

Pitfalls

Pitfalls of benchmarking

- Intra- and inter-runs variance is variable between benchmarks;
- Hitting the power budget, a thermal limit or GPU reset;
- Being able to reproduce the different test results;
- Not using the expected libraries;
- Comparing runs generated using a different environment:
  - Kernel, libdrm and mesa's version and config;
  - Display server used (and its configuration);
  - Hardware and BIOS versions.

Pitfalls - Intra- and inter-run variance

The variance forces us to execute multiple runs, which takes time!

Intra-run variance due to

- Power management (Boost-like features, thermal throttling);
- Concurrent tasks generating IOs, CPU or GPU load;
- Interrupts from the hardware;
- CPU/GPU schedulers.

Inter-run variance due to

- Variations in the memory allocation.

Pitfalls - Recommendations to reduce the variance

CPU-limited cases

- Force the CPU to one single frequency;
- Pin the game/benchmark to a single core;
- Disable ASLR and transparent huge pages;
- Run as little services as possible;
- Pin IRQs to another core;
- Properly cool the device.

GPU-limited cases

- Force the GPU to one frequency;
- Reduce the number of active GPU contexts;
- Properly cool the device.

Pitfalls - Recommendations to reduce the variance

Problem

If we change the environment, we skew the results!

Be smart!

Only get rid of what you are not trying to optimise or track!

Many variables to check, track and remember!

We need to help the developers and QA by automating all we can!

Automated benchmarking

Objectives of automated benchmarking

- Avoid or detect human errors;
- Make sure the data is valid;
- Be predictable in the execution time;
- Provide as much information as possible;
- Guarantee reproducibility of the results.

In concrete goals

- Be aware of every library used by the program;
- Know their versions, git ID and compilation flags;
- Poll on the resources' usage metrics;
- Store all this information inside the report;
- Understand performance results and act upon them.

Automated benchmarking - Making sure the data is valid

Making sure the data is valid

- Compute the statistical accuracy and add runs if needed;
- Get information out from the kernel about major hw events;
- Learn to give up and re-prioritise other benchmarks;
- Try to reproduce runs and detect major differences;
- Reboot the machine if unsure about the results;
- Collect usage metrics of the resources;
- Log all this information in the report.

Bisect performance changes automatically

- It adds credibility to the report;
- It also reproduces the issue.

Automated benchmarking - Guaranteeing reproducibility

Guaranteeing reproducibility - Why?

- Allow developers to reproduce a performance regression.

Challenges

- How do we detect the entire environment of the benchmark?

Automated benchmarking - Reading out the environment

Listing dependencies

- Using ldd is insufficient because of run-time dependencies;
- Strace is the most robust approach but it is slow;
- Linked libraries can be listed in /proc/pid/maps.

Query the version of a library/program

- No silver bullet;
- Can sometimes be read out of a program (Linux);
- Will often require controlling the build process.

Automated benchmarking - What tools?

Phoronix Testing Suite - Pros

- Automates data acquisition;
- Collects some useful metrics.

Phoronix Testing Suite - Cons

- Oriented towards simple reporting, no good for performance analysts;
- Reads out the environment but with no guarantees;
- Hides performance data;
- Not git-centric.

Automated benchmarking - What tools?

Ben Widawsky's tool - Pros

- Strong modelling effort to validate the reported values;
- Detects some hardware events and invalidates data;
- Great for developer experiments, not for QA.

Ben Widawsky's tool - Cons

- Non-build- and non-git-aware;
- Not aware of the environment;
- Supports a limited amount of benchmarks;
- Requires a lot of manual work to test big series.

Automated benchmarking - What tools?

Ideal system

Manages the build system, the commit history and the environment of benchmarks while allowing metrics collection. Should provide a visual report that eases performance analysis.

Summary

EzBench - Overview

Ezbench - Goals

- Provide workflows and automation to take care of most issues;
- Provide a framework quickly adaptable to your needs;
- Work for both QA and developers!

Authors

- Authors: Martin Peres (Intel) & Chris Wilson (Intel);
- Licence: MIT;
- Url: http://cgit.freedesktop.org/~mperes/ezbench/

EzBench - Features

Current features

- Modular architecture (profiles, tests and user hooks);
- Automates the acquisition of benchmark data;
- Generates a report that is usable by developers;
- Bisects performance changes automatically;
- Provides python bindings to acquire data and parse them.

WIP

- Be crash-resistant by storing the expected goal and comparing it to the current state;
- Use a modelling approach to detect performance changes;
- Detect the environment.

EzBench - Features

TODO

- Detect HW events and react to them;
- Predict run times more accurately;
- Support deadlines and test prioritisation;
- Support sending emails to the authors of perf changes;
- Integrate with patchwork to test patch series.

EzBench - Demo time!

Demo time and questions!