EzBench, a tool to help you benchmark and bisect the Graphics Stack's performance

Martin Peres

Intel Open Source Technology Center Finland

September 19, 2016

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.

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

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.

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

Summary

Benchmarking

Different needs for benchmarking

- Developers: Run multiple experiments and compare them;

Benchmarking

Different needs for benchmarking

- Developers: Run multiple experiments and compare them;
- QA: Continuous Integration, performance bug reports.

Pitfalls

Pitfalls of benchmarking

- Intra- and inter-runs variance depends on the benchmarks;

Pitfalls

Pitfalls of benchmarking

- Intra- and inter-runs variance depends on the benchmarks;
- Hitting the power budget, a thermal limit or GPU reset;

Pitfalls

Pitfalls of benchmarking

- Intra- and inter-runs variance depends on the benchmarks;
- Hitting the power budget, a thermal limit or GPU reset;
- Being able to reproduce the different test results;

Pitfalls

Pitfalls of benchmarking

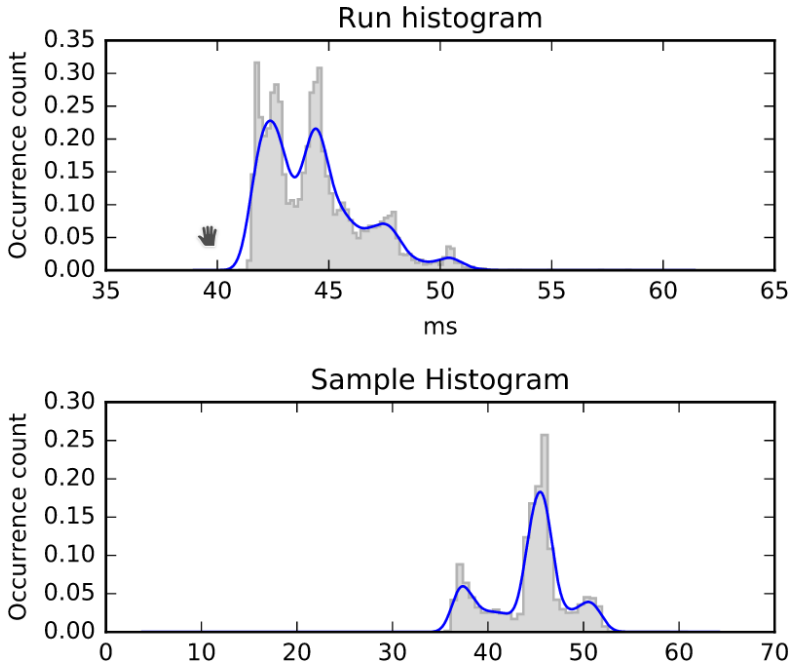
- Intra- and inter-runs variance depends on the benchmarks;
- Hitting the power budget, a thermal limit or GPU reset;
- Being able to reproduce the different test results;
- Not using the expected libraries;

Example of variances

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

Example of variances

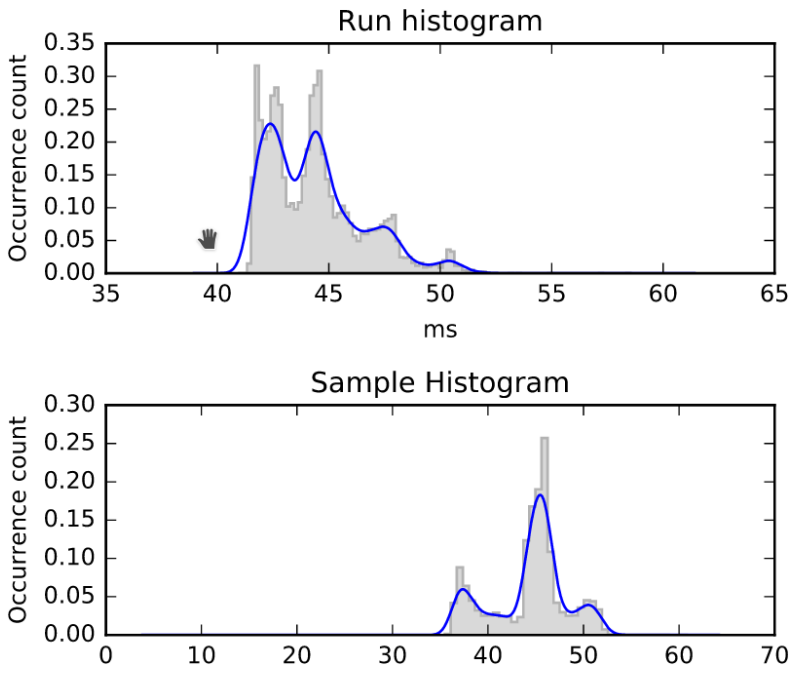
The variance forces us to execute multiple runs, which takes time!



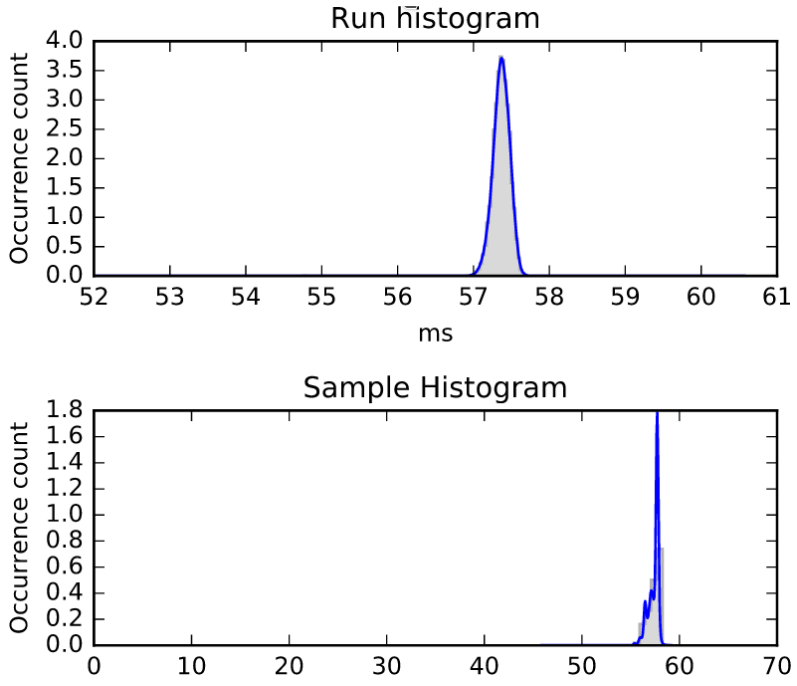
(a) Bad FPS distribution

Example of variances

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



(a) Bad FPS distribution



(b) Good FPS distribution

Figure: Examples of variance

Automated benchmarking

Objectives of automated benchmarking

- Avoid or detect human errors;

Automated benchmarking

Objectives of automated benchmarking

- Avoid or detect human errors;
- Make sure the data is valid;

Automated benchmarking

Objectives of automated benchmarking

- Avoid or detect human errors;
- Make sure the data is valid;
- Be predictable in the execution time;

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;

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.

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;

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;

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;

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 a report

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 a 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;

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;

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;

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;

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;

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;

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.

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;

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 - Reading out the environment

Listing dependencies

- Using ldd is insufficient because of run-time dependencies;

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;

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 polled from /proc/$pid/maps;

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 polled from /proc/$pid/maps;
- We can hook some functions using LD_PRELOAD.

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 polled from /proc/$pid/maps;
- We can hook some functions using LD_PRELOAD.

Query the version of a library/program

- No silver bullet;

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 polled from /proc/$pid/maps;
- We can hook some functions using LD_PRELOAD.

Query the version of a library/program

- No silver bullet;
- Can sometimes be read out of a program (Linux);

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 polled from /proc/$pid/maps;
- We can hook some functions using LD_PRELOAD.

Query the version of a library/program

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

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 polled from /proc/$pid/maps;
- We can hook some functions using LD_PRELOAD.

Query the version of a library/program

- No silver bullet;
- Can sometimes be read out of a program (Linux);
- Requires controlling the build process;
- Requires package-kit for system libraries.

Summary

EzBench - Overview

Ezbench - Goals

- Provide workflows and automation to take care of most issues;

EzBench - Overview

Ezbench - Goals

- Provide workflows and automation to take care of most issues;
- Provide a framework quickly adaptable to your needs;

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!

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 - Components

Components

- core.sh: simple runner;

EzBench - Components

Components

- core.sh: simple runner;
- env_dump: dump environment;

EzBench - Components

Components

- core.sh: simple runner;
- env_dump: dump environment;
- ezbench: work scheduler;

EzBench - Components

Components

- core.sh: simple runner;
- env_dump: dump environment;
- ezbench: work scheduler;
- utils/ezbench.py: framework;

EzBench - Components

Components

- core.sh: simple runner;
- env_dump: dump environment;
- ezbench: work scheduler;
- utils/ezbench.py: framework;
- utils/ezbenchd.py: work executer;

EzBench - Components

Components

- core.sh: simple runner;
- env_dump: dump environment;
- ezbench: work scheduler;
- utils/ezbench.py: framework;
- utils/ezbenchd.py: work executer;
- stats/compare_reports.py: visualisation.

EzBench - SHA1-DB

SHA1-DB

- Stores SHA1 hashes of the libs you compile;

EzBench - SHA1-DB

SHA1-DB

- Stores SHA1 hashes of the libs you compile;
- Allows you to attach metadata to the hash:

EzBench - SHA1-DB

SHA1-DB

- Stores SHA1 hashes of the libs you compile;
- Allows you to attach metadata to the hash:
  - Git commit SHA1;

EzBench - SHA1-DB

SHA1-DB

- Stores SHA1 hashes of the libs you compile;
- Allows you to attach metadata to the hash:
  - Git commit SHA1;
  - Compilation flags;

EzBench - SHA1-DB

SHA1-DB

- Stores SHA1 hashes of the libs you compile;
- Allows you to attach metadata to the hash:
  - Git commit SHA1;
  - Compilation flags;
  - Whatever you want!

EzBench - Env Dump

Env Dump

- Shared object LD_PRELOADed when running benchmarks;

EzBench - Env Dump

Env Dump

- Shared object LD_PRELOADed when running benchmarks;
- Captures information about:

EzBench - Env Dump

Env Dump

- Shared object LD_PRELOADed when running benchmarks;
- Captures information about:
  - HW topology (CPU, RAM, BIOS, MOTHERBOARD);

EzBench - Env Dump

Env Dump

- Shared object LD_PRELOADed when running benchmarks;
- Captures information about:
  - HW topology (CPU, RAM, BIOS, MOTHERBOARD);
  - Dependencies to libraries, binaries and UNIX services;

EzBench - Env Dump

Env Dump

- Shared object LD_PRELOADed when running benchmarks;
- Captures information about:
  - HW topology (CPU, RAM, BIOS, MOTHERBOARD);
  - Dependencies to libraries, binaries and UNIX services;
  - X interactions (window/screen sizes);

EzBench - Env Dump

Env Dump

- Shared object LD_PRELOADed when running benchmarks;
- Captures information about:
  - HW topology (CPU, RAM, BIOS, MOTHERBOARD);
  - Dependencies to libraries, binaries and UNIX services;
  - X interactions (window/screen sizes);
  - GL/GLX/EGL contexts;

EzBench - Env Dump

Env Dump

- Shared object LD_PRELOADed when running benchmarks;
- Captures information about:
  - HW topology (CPU, RAM, BIOS, MOTHERBOARD);
  - Dependencies to libraries, binaries and UNIX services;
  - X interactions (window/screen sizes);
  - GL/GLX/EGL contexts;
  - Environment variables.

EzBench - Demo time!

Demo time and questions!

EzBench - Features

Current features

- Modular architecture (profiles, tests and user hooks);

EzBench - Features

Current features

- Modular architecture (profiles, tests and user hooks);
- Automates the acquisition of benchmark data;

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;

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;

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 reports;

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 reports;
- Be crash-resistant by storing the expected goal and comparing it to the current state;

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 reports;
- Be crash-resistant by storing the expected goal and comparing it to the current state;
- Collect the environment information and diff it;

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 reports;
- Be crash-resistant by storing the expected goal and comparing it to the current state;
- Collect the environment information and diff it;
- Detect the variance and peformance changes;

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 reports;
- Be crash-resistant by storing the expected goal and comparing it to the current state;
- Collect the environment information and diff it;
- Detect the variance and peformance changes;
- Automatically schedule more work to improve the report.

EzBench - Features

TODO

- Watchdog support;

EzBench - Features

TODO

- Watchdog support;
- Handle kernel boot failures;

EzBench - Features

TODO

- Watchdog support;
- Handle kernel boot failures;
- Add support for PTS as a backend;

EzBench - Features

TODO

- Watchdog support;
- Handle kernel boot failures;
- Add support for PTS as a backend;
- Better integrate the build process;

EzBench - Features

TODO

- Watchdog support;
- Handle kernel boot failures;
- Add support for PTS as a backend;
- Better integrate the build process;
- React to HW events such as throttling;

EzBench - Features

TODO

- Watchdog support;
- Handle kernel boot failures;
- Add support for PTS as a backend;
- Better integrate the build process;
- React to HW events such as throttling;
- Reset the environment to a previous state;

EzBench - Features

TODO

- Watchdog support;
- Handle kernel boot failures;
- Add support for PTS as a backend;
- Better integrate the build process;
- React to HW events such as throttling;
- Reset the environment to a previous state;
- Integrate with patchwork to test patch series;

EzBench - Features

TODO

- Watchdog support;
- Handle kernel boot failures;
- Add support for PTS as a backend;
- Better integrate the build process;
- React to HW events such as throttling;
- Reset the environment to a previous state;
- Integrate with patchwork to test patch series;
- Predict run times more accurately (compilation done);

EzBench - Features

TODO

- Watchdog support;
- Handle kernel boot failures;
- Add support for PTS as a backend;
- Better integrate the build process;
- React to HW events such as throttling;
- Reset the environment to a previous state;
- Integrate with patchwork to test patch series;
- Predict run times more accurately (compilation done);
- Support sending emails to the authors of perf changes.