Very fast code profiling with tracy

Wilson Jallet

Background: profiling code

- identify performance **bottlenecks**: memory allocation, recurring function calls
- measure time of functions in the **call stack**

On Linux systems: *perf* kernel tool: https://perf.wiki.kernel.org

- Visualizing with external tools such as **hotspot:** github.com/KDAB/hotspot
- Works by **event-based sampling**. Limited by its sampling frequency.
- Also supports annotating source code.
- Some counters exist at the hardware level.

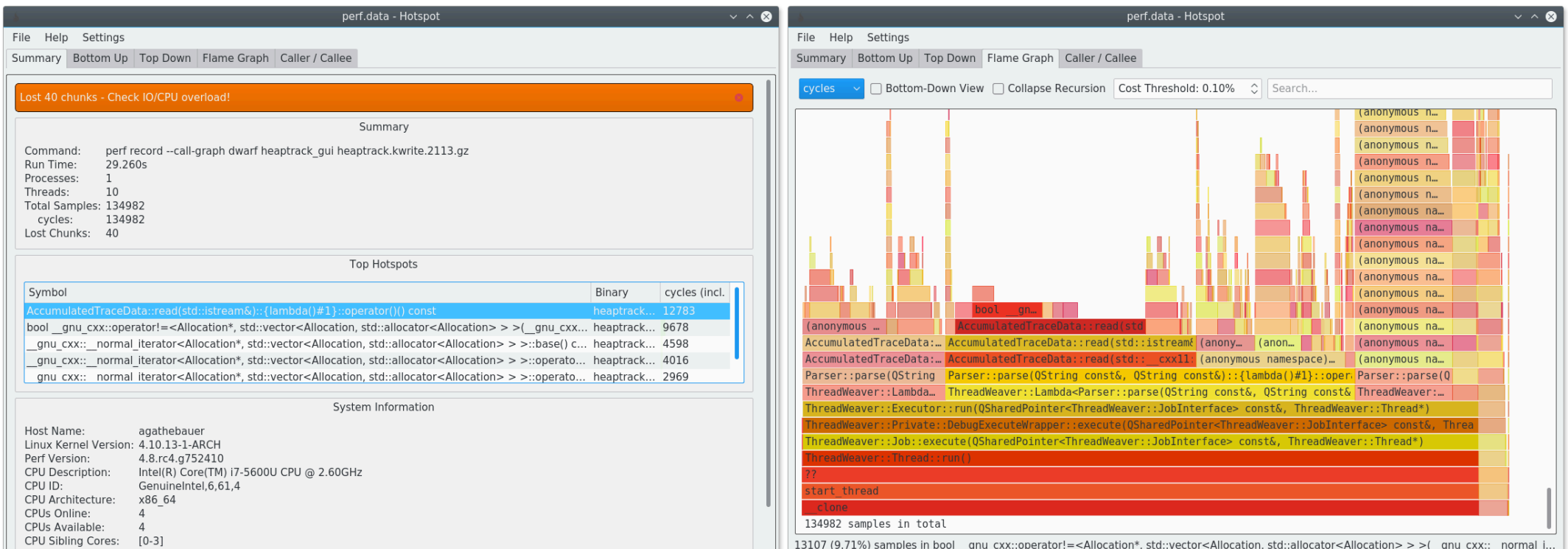

Figure 1: The **hotspot** GUI for *perf*. It provides an overview of function calls, flamegraphs, and more.

A modern, cross-platform alternative: *Tracy*.

- emphasis on manually annotating code (sampling also possible)
- comes with a sleek real-time UI
- mainly C++, also has C API and Rust bindings

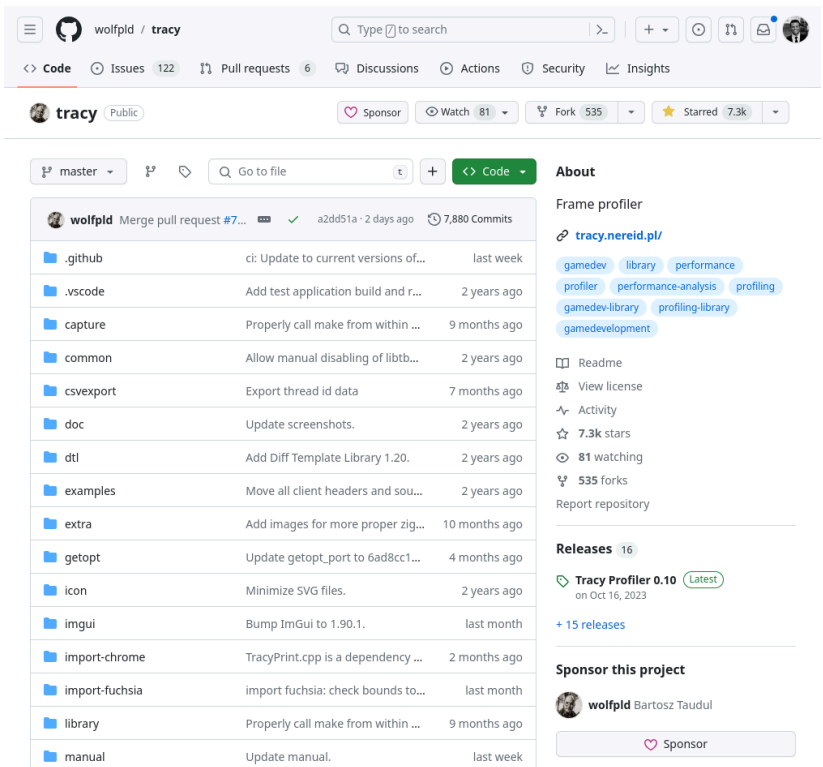

Figure 2: Tracy's GitHub page

Tracy has become a popular industry choice:

- mostly videogame studios (emphasis on *frame delays*)
- very precise measurement of function calls (down to the **nanosecond**)
- even works with Vulkan, OpenGL, DX11/DX12 calls...

on overdrive. Here's a short (and incomplete) list of companies and projects which already use Tracy and find it useful:

- QLOC (Mortal Kombat 11)
- Bohemia Interactive (Arma 3, Reforger)
- Huuuge Games (Huuuge Casino)
- Artifex Mundi (Endless Fables 4)
- SCS Software (Euro/American Truck Simulator)
- CD Projekt Red (Cyberpunk 2077)
- Unknown Worlds (Natural Selection 2)
- EXOR Studios (The Riftbreaker)
- CERN (ALICE experiment)
- Adobe (Oculus Medium, Shaper)
- NVIDIA (Omniverse)
- Amazon (Lumberyard)
- Mozilla (webrender)
- Frontier Developments (COBRA engine)
- Blizzard (Diablo Immortal)
- Gameloft
- Fatshark (https://www.fatshark.se/)
- VT MAK (https://www.mak.com/)
- Allplan (https://www.allplan.com/)
- OpenSpace (https://www.openspaceproject.com/)
- Flagship Biosciences (proprietary microscope software)
- DigiPen Institute of Technology Singapore (game development course)
- Cocaine Diesel game (https://cocainediesel.fun/)
- nCine game engine (https://ncine.github.io/)
- Deepkit (https://deepkit.ai/)
- Polystream (cloud gaming service)
- Newton Dynamics (http://newtondynamics.com/)
- Codeplay Software (ComputeCpp)
- Tanvas (https://tanvas.co/)
- TomTom (Peugeot Landtrek)
- The Dark Mod (https://www.thedarkmod.com/)
- Netflix
- Boston Dynamics

Tracy works by **manually instrumenting your code**. In practice, adding macro calls in your code:

```
#include <tracy/Tracy.hpp> // adds #define ZoneScoped
#include <vector>
#include <algorithm>

double compute_sum(std::vector<double> &vec) {
  ZoneScoped;
  return std::accumulate(vec.begin(), vec.end(), 0);
}
```

Now, calls to `compute_sum()` will show up in the Tracy profiler GUI!

webrender/src/renderer/mod.rs — Rust · master

```
1279        // event. Otherwise they would just pile up in this vector forever.
1280        self.notifications.clear();
1281
1282        tracy_frame_marker!();
1283
1284        result
1285    }
```

webrender/src/scene_builder_thread.rs — Rust · master

```
304         loop {
305             tracy_begin_frame!("scene_builder_thread");
306
393
394             tracy_end_frame!("scene_builder_thread");
395         }
```

Figure 4: Use of Tracy in Rust (from the source of *servo*, Firefox's web renderer)

Tracy provides a convenient web demo:

https://tracy.nereid.pl/

and an amazing & informative user manual

https://github.com/wolfpld/tracy/releases/latest/download/tracy.pdf

In conclusion:

There already are good classic tools for profiling e.g. *perf*:

- might be a bit obscure
- not crossplatform

Tracy provides

- cross-platform and device support
- very nice analysis tools
- nanosecond-level insight through annotated code