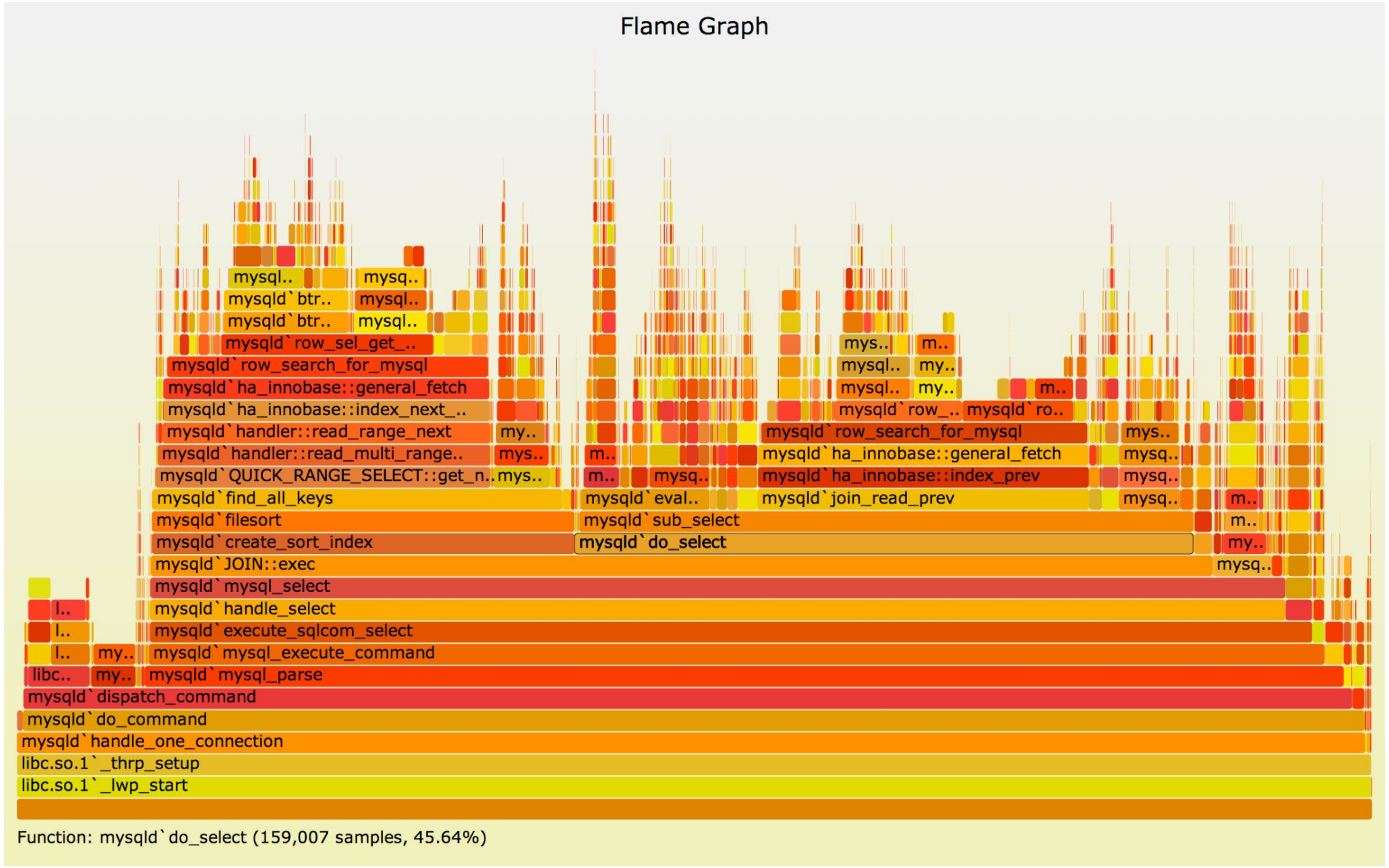
Visualizing Execution with FlameGraphs

What you learn in the this session

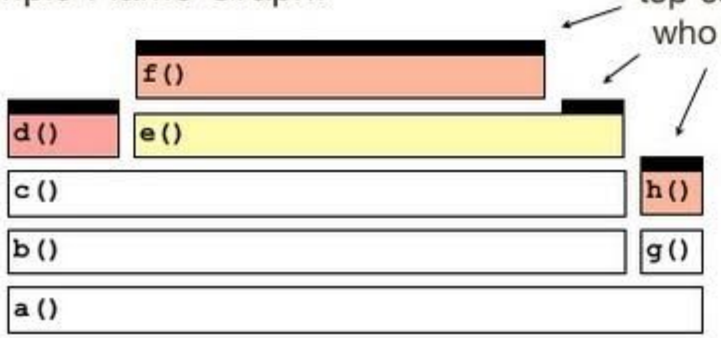
- What FlameGraphs are and how to read them
- How to generate them
- What the performance impact is on supporting them
- How to generate them from inside a container


Flame Graph
mysql..
mysq..
mysqld`btr..
mysql..
mysqld`btr..
mysql..
mysqld`row_sel_get_..
mysqld`row_search_for_mysql
mysqld`ha_innobase::general_fetch
mysqld`ha_innobase::index_next_..
mysqld`handler::read_range_next
my..
mysqld`handler::read_multi_range..
mys..
mysqld`QUICK_RANGE_SELECT::get_n..
mys..
mysqld`find_all_keys
mysqld`filesort
mysqld`create_sort_index
mysqld`JOIN::exec
mysqld`mysql_select
mysqld`handle_select
mysqld`execute_sqlcom_select
mysqld`mysql_execute_command
mysqld`mysql_parse
mysqld`dispatch_command
mysqld`do_command
mysqld`handle_one_connection
libc.so.1`_thrp_setup
libc.so.1`_lwp_start
l..
l..
l..
my..
libc..
my..
mys..
m..
mysql..
my..
mysql..
my..
mysqld`row_..
mysqld`ro..
m..
mysqld`row_search_for_mysql
mys..
m..
mysqld`ha_innobase::general_fetch
mysq..
m..
mysq..
mysqld`ha_innobase::index_prev
mysq..
mysqld`eval..
mysqld`join_read_prev
mysq..
m..
mysqld`sub_select
m..
mysqld`do_select
my..
mysq..
Function: mysqld`do_select (159,007 samples, 45.64%)

- Invented by Brendan Gregg (Sun, Oracle, Netflix, Intel) 2011
    - Presented at LISA 2013 https://www.brendangregg.com/Slides/LISA13_Flame_Graphs.pdf
- Picture of relative time spent in a single image
- Visual, navigable, obvious
    - https://raw.githubusercontent.com/brendangregg/FlameGraph/master/example-perf.svg
- Low overhead (you will measure this!)

Flame Graphs: How to Read

- A CPU Sample Flame Graph:

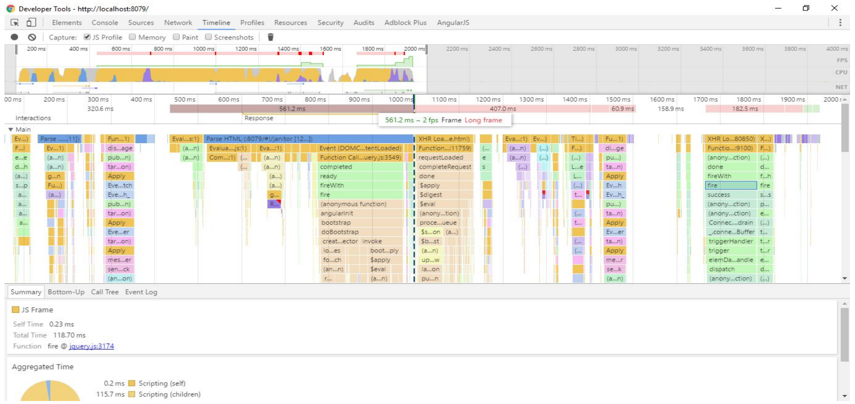


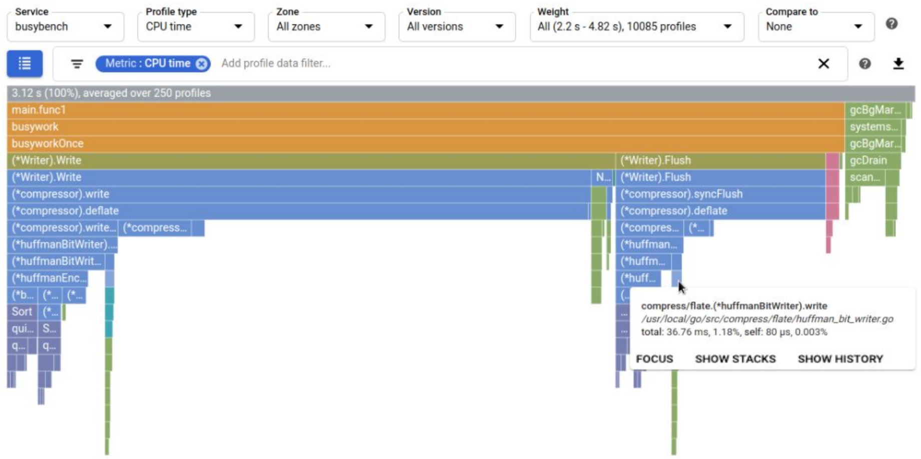
top edge shows who is on-CPU directly

- Q: which function is on-CPU the most?
- A: `f()`

`e()` is on-CPU a little, but its runtime is mostly spent in `f()`, which is on-CPU directly

https://netflixtechblog.com/java-in-flames-e763b3d32166


Developer Tools - http://localhost:8079/
Elements Console Sources Network Timeline Profiles Resources Security Audits Adblock Plus AngularJS
Capture: JS Profile Memory Paint Screenshots
Summary Bottom-Up Call Tree Event Log
JS Frame
Self Time 0.23 ms
Total Time 118.70 ms
Function fire @ jquery.js:3174
Aggregated Time
0.2 ms Scripting (self)
115.7 ms Scripting (children)


Service: busybench
Profile type: CPU time
Zone: All zones
Version: All versions
Weight: All (2.2 s - 4.82 s), 10085 profiles
Compare to: None

Metric : CPU time
Add profile data filter...

3.12 s (100%), averaged over 250 profiles

compress/flate.(*huffmanBitWriter).write
/usr/local/go/src/compress/flate/huffman_bit_writer.go
total: 36.76 ms, 1.18%, self: 80 µs, 0.003%

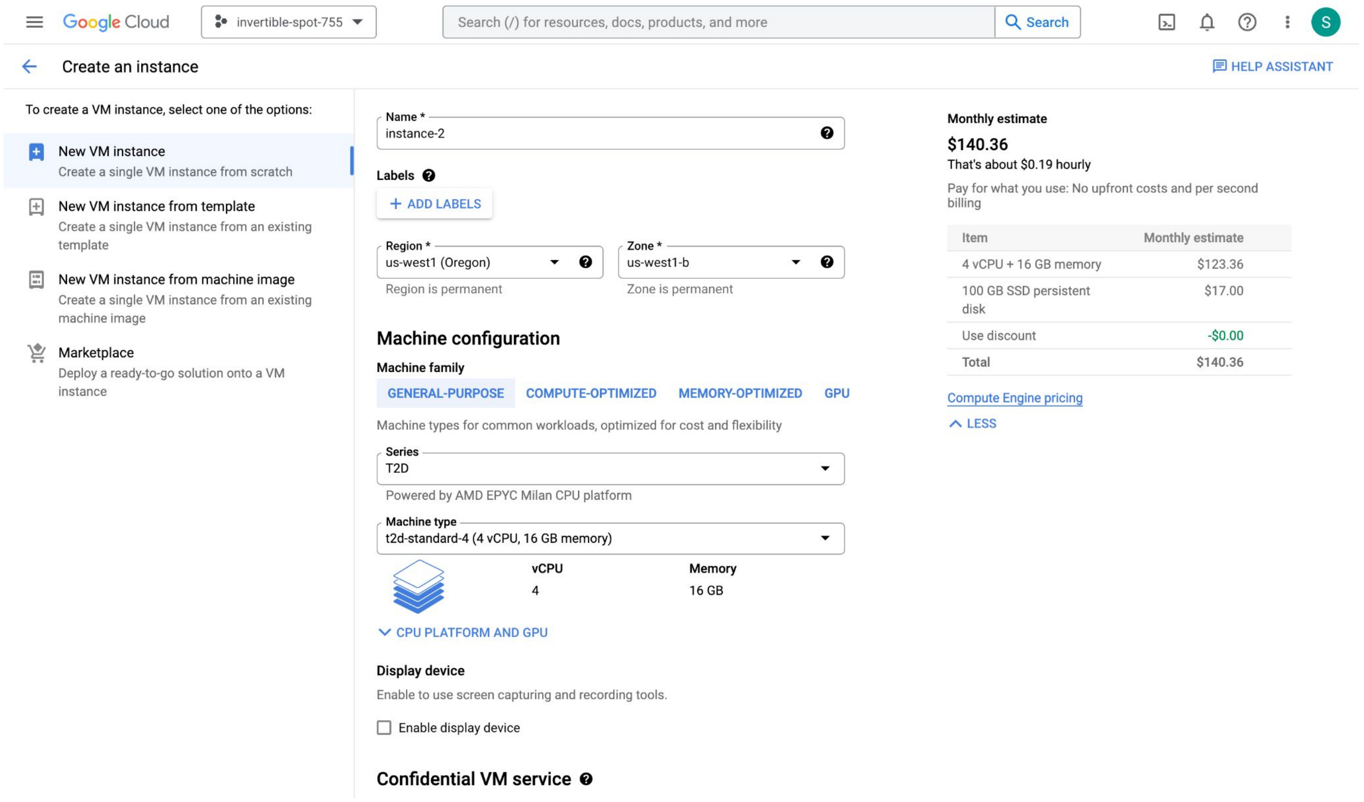
- Broad Industry Support
  - Chrome Devtools FlameCharts* (x axis is time, not aggregated counts)
    - https://developer.chrome.com/docs/devtools/performance/reference/
  - Google Cloud Profiler (IcicleGraph), rendered from the top down
    - https://cloud.google.com/profiler/docs/measure-app-performance
    - https://cloud.google.com/profiler/docs/concepts-flame
      - Built-in support for Java, Python, Node and Go
  - Grafana
    - Continuous Profiling Tool https://github.com/grafana/phlare (easily deploys on K8S)
  - Supported on everywhere
    - Windows, OSX, FreeBSD
  - Supported in every popular language

Great Documentation

- USENIX ATC '17: Visualizing Performance with Flame Graphs
  - https://www.youtube.com/watch?v=D53T1Ejig1Q
- LISA13 - Blazing Performance with Flame Graphs
  - https://www.youtube.com/watch?v=nZfNehCzGdw
- https://github.com/brendangregg/FlameGraph
- https://www.brendangregg.com/flamegraphs.html

- Stack Sampling, sample as often or as little as your application dictates
  - `perf record -F 100 -g -- <application under measurement>`
- Can be generated by **any** profiler that captures stack back traces
  - Perf, Dtrace, Instruments, SystemTap, VTune, ktap, Xperf
  - Better quality stack information, better FlameGraphs compile binaries with
    - `-g -fno-omit-frame-pointer`
    - Fedora enabling by default in 38
      - https://fedoraproject.org/wiki/Changes/fno-omit-frame-pointer
      - Brendan Gregg wants on by default in GCC
- Install Linux perf command
  - `apt install linux-tools-$(uname -r) linux-tools-common`
  - When running in a container, tools in container must match host kernel!!

Create a VM on the Google Cloud Console



Create an instance

To create a VM instance, select one of the options:

- **New VM instance** – Create a single VM instance from scratch
- New VM instance from template – Create a single VM instance from an existing template
- New VM instance from machine image – Create a single VM instance from an existing machine image
- Marketplace – Deploy a ready-to-go solution onto a VM instance

Name *: instance-2

Labels

+ ADD LABELS

Region *: us-west1 (Oregon) — Region is permanent

Zone *: us-west1-b — Zone is permanent

Machine configuration

Machine family: GENERAL-PURPOSE | COMPUTE-OPTIMIZED | MEMORY-OPTIMIZED | GPU

Machine types for common workloads, optimized for cost and flexibility

Series: T2D

Powered by AMD EPYC Milan CPU platform

Machine type: t2d-standard-4 (4 vCPU, 16 GB memory)

vCPU	Memory
4	16 GB

CPU PLATFORM AND GPU

Display device

Enable to use screen capturing and recording tools.

☐ Enable display device

Confidential VM service

Monthly estimate

$140.36

That's about $0.19 hourly

Pay for what you use: No upfront costs and per second billing

Item	Monthly estimate
4 vCPU + 16 GB memory	$123.36
100 GB SSD persistent disk	$17.00
Use discount	-$0.00
Total	$140.36

Compute Engine pricing

LESS

VM Recommendations

- Ubuntu 22.04 LTS
- t2d-standard-4 (4 physical cores, 16GB)
- 20GB+ PD-SSD for boot disk (IOPs scale with disk size)
- Allow HTTP and HTTPS (only for zero risk, ephemeral VMs)

First Exercise

- Follow the README in this repo
  https://github.com/jensengrey/flamegraph-container

Second Exercise

- Same VM or new VM
  - Take a VM Snapshot?
  - Brand new VM?
- Building and running two programs
  - Small Obfuscated Ray Tracer
  - POV-Ray

Prepare your VM

```
# apt based (Ubuntu/Debian)
sudo sh -c 'apt update -y && apt upgrade -y'
sudo apt install build-essential -y
sudo apt install make tmux tree htop imagemagick nano vim git cmake -y
sudo apt install python3-dev python3-venv

# homebrew, https://brew.sh/
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# rust, https://rustup.rs/
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Create Python Environment

```
#!/bin/bash

set -eux;

python3 -m venv test.env
source test.env/bin/activate
pip install -U pip setuptools wheel
```

Install Perf Command

```
sudo apt install linux-tools-$(uname -r) linux-tools-common -y
```

Install FlameGraph Tools

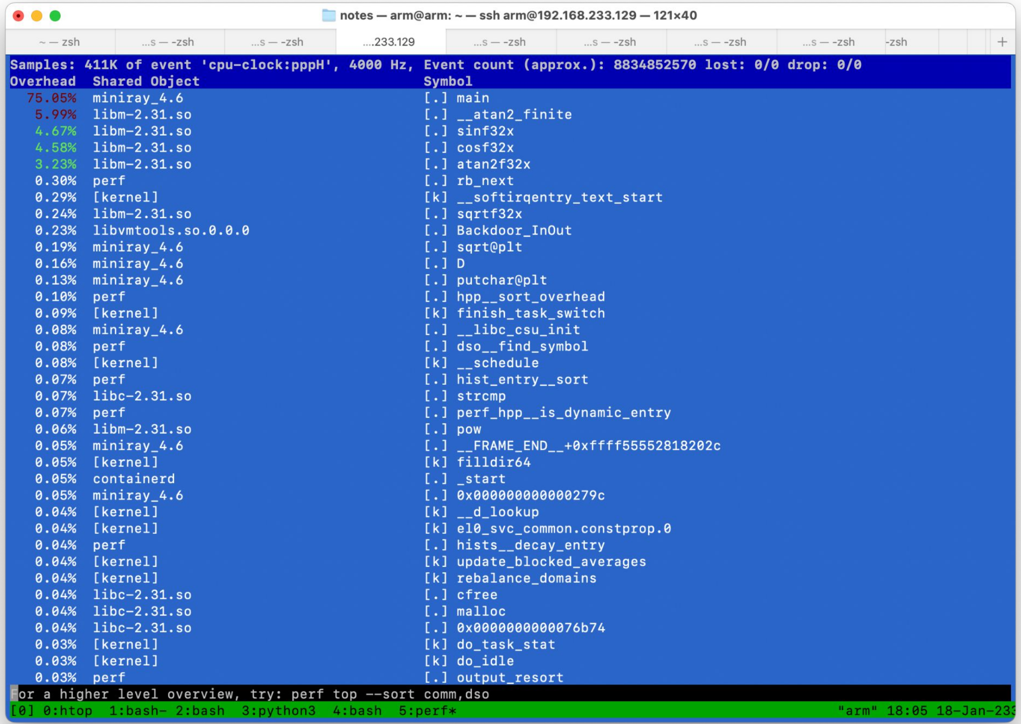
```
# the canonical tool for converting perf traces to svg
git clone https://github.com/brendangregg/FlameGraph

# more performant in some scenarios
```

- https://github.com/flamegraph-rs/flamegraph
    - https://github.com/jonhoo/inferno

Enable sudo-less event collection

```
sudo sh -c "echo 0 > /proc/sys/kernel/kptr_restrict"
sudo sh -c "echo -1 > /proc/sys/kernel/perf_event_paranoid"
```

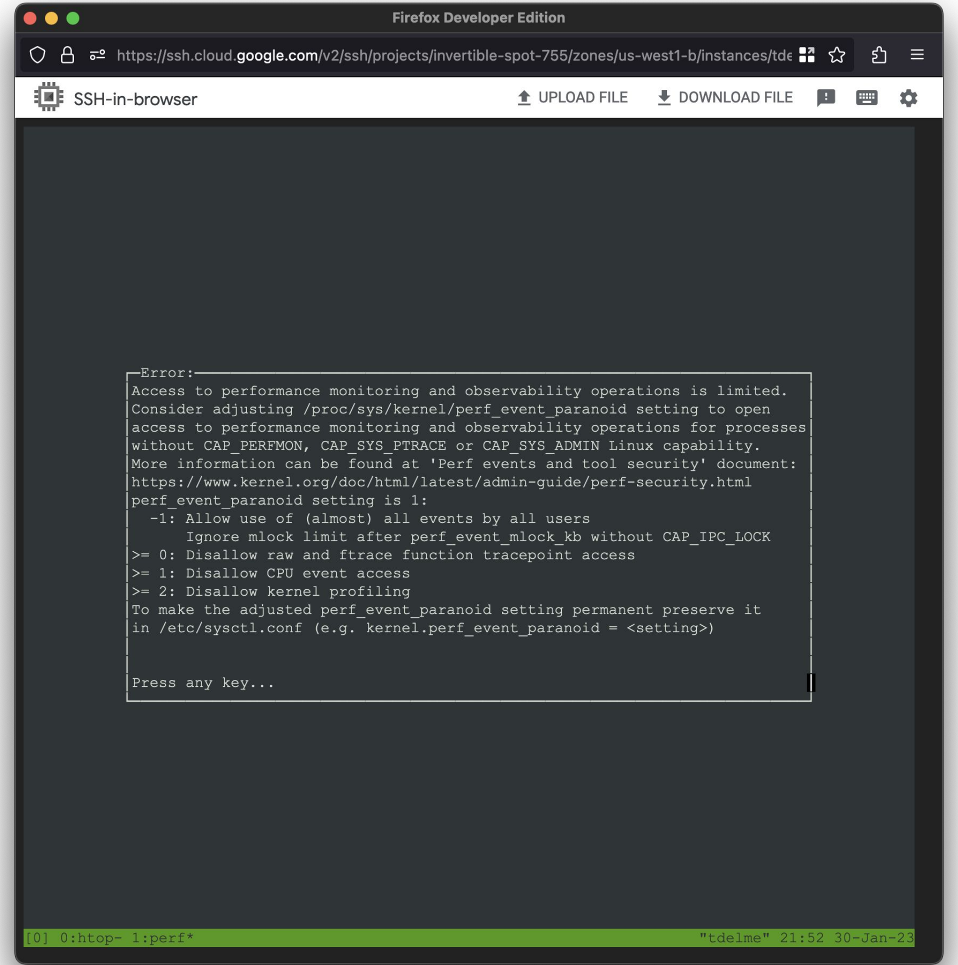
perf top -e cpu-clock



```
Samples: 411K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 8834852570 lost: 0/0 drop: 0/0
Overhead  Shared Object                             Symbol
  75.05%  miniray_4.6                               [.] main
   5.99%  libm-2.31.so                              [.] __atan2_finite
   4.67%  libm-2.31.so                              [.] sinf32x
   4.58%  libm-2.31.so                              [.] cosf32x
   3.23%  libm-2.31.so                              [.] atan2f32x
   0.30%  perf                                      [.] rb_next
   0.29%  [kernel]                                  [k] __softirqentry_text_start
   0.24%  libm-2.31.so                              [.] sqrtf32x
   0.23%  libvmtools.so.0.0.0                       [.] Backdoor_InOut
   0.19%  miniray_4.6                               [.] sqrt@plt
   0.16%  miniray_4.6                               [.] D
   0.13%  miniray_4.6                               [.] putchar@plt
   0.10%  perf                                      [.] hpp__sort_overhead
   0.09%  [kernel]                                  [k] finish_task_switch
   0.08%  miniray_4.6                               [.] __libc_csu_init
   0.08%  perf                                      [.] dso__find_symbol
   0.08%  [kernel]                                  [k] __schedule
   0.07%  perf                                      [.] hist_entry__sort
   0.07%  libc-2.31.so                              [.] strcmp
   0.07%  perf                                      [.] perf_hpp__is_dynamic_entry
   0.06%  libm-2.31.so                              [.] pow
   0.05%  miniray_4.6                               [.] __FRAME_END__+0xffff55552818202c
   0.05%  [kernel]                                  [k] filldir64
   0.05%  containerd                                [.] _start
   0.05%  miniray_4.6                               [.] 0x000000000000279c
   0.04%  [kernel]                                  [k] __d_lookup
   0.04%  [kernel]                                  [k] el0_svc_common.constprop.0
   0.04%  perf                                      [.] hists__decay_entry
   0.04%  [kernel]                                  [k] update_blocked_averages
   0.04%  [kernel]                                  [k] rebalance_domains
   0.04%  libc-2.31.so                              [.] cfree
   0.04%  libc-2.31.so                              [.] malloc
   0.04%  libc-2.31.so                              [.] 0x0000000000076b74
   0.03%  [kernel]                                  [k] do_task_stat
   0.03%  [kernel]                                  [k] do_idle
   0.03%  perf                                      [.] output_resort
For a higher level overview, try: perf top --sort comm,dso
[0] 0:htop  1:bash- 2:bash  3:python3  4:bash  5:perf*                                               "arm" 18:05 18-Jan-23
```

What you see when perf is restricted

```
sudo sh -c "echo 1 > /proc/sys/kernel/kptr_restrict"
sudo sh -c "echo 1 > /proc/sys/kernel/perf_event_paranoid"
```


Firefox Developer Edition
https://ssh.cloud.google.com/v2/ssh/projects/invertible-spot-755/zones/us-west1-b/instances/tde
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE

```
┌Error:──────────────────────────────────────────────────────────────────────┐
│Access to performance monitoring and observability operations is limited.   │
│Consider adjusting /proc/sys/kernel/perf_event_paranoid setting to open     │
│access to performance monitoring and observability operations for processes│
│without CAP_PERFMON, CAP_SYS_PTRACE or CAP_SYS_ADMIN Linux capability.      │
│More information can be found at 'Perf events and tool security' document: │
│https://www.kernel.org/doc/html/latest/admin-guide/perf-security.html       │
│perf_event_paranoid setting is 1:                                           │
│  -1: Allow use of (almost) all events by all users                         │
│      Ignore mlock limit after perf_event_mlock_kb without CAP_IPC_LOCK     │
│>= 0: Disallow raw and ftrace function tracepoint access                    │
│>= 1: Disallow CPU event access                                             │
│>= 2: Disallow kernel profiling                                             │
│To make the adjusted perf_event_paranoid setting permanent preserve it      │
│in /etc/sysctl.conf (e.g. kernel.perf_event_paranoid = <setting>)           │
│                                                                            │
│                                                                            │
│Press any key...                                                            │
└────────────────────────────────────────────────────────────────────────────┘
[0] 0:htop- 1:perf*                                  "tdelme" 21:52 30-Jan-23
```

Enable sudo-less event collection

```
sudo sh -c "echo 0 > /proc/sys/kernel/kptr_restrict"
sudo sh -c "echo -1 > /proc/sys/kernel/perf_event_paranoid"
```

Call graph recording

Samples per second

```
perf record -F 100 -a -g -- <application under measurement>
```

All cores

First Test Workload

```
git clone https://github.com/mzucker/miniray
cd miniray
mkdir build
```

Edit ../CmakeLists.txt

```
cmake_minimum_required(VERSION 2.8.12)

set(GCC_COVERAGE_COMPILE_FLAGS "-g3 -fno-omit-frame-pointer")

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${GCC_COVERAGE_COMPILE_FLAGS}")
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${GCC_COVERAGE_COMPILE_FLAGS}")

project(miniray)

set(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR})

add_subdirectory(src)
```

Run cmake

```
 cmake ..
-- The C compiler identification is GNU 11.3.0
-- The CXX compiler identification is GNU 11.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/sean_jensengrey/miniray/build
```

Build the code

```
$ make
[  3%] Built target encode
[  6%] Built target miniray_4.6
[ 10%] Built target miniray_4.5
[ 13%] Built target miniray_1.3
[ 16%] Built target miniray_4.6_commented
[ 20%] Built target miniray_4.0
[ 23%] Built target miniray_2.4
...
# src/miniray_0.2.cpp:73:3: error: narrowing conversion of ‘2169135176’ from
‘unsigned int’ to ‘i’ {aka ‘int’} [-Wnarrowing]
```

:) "Working version" of that Code

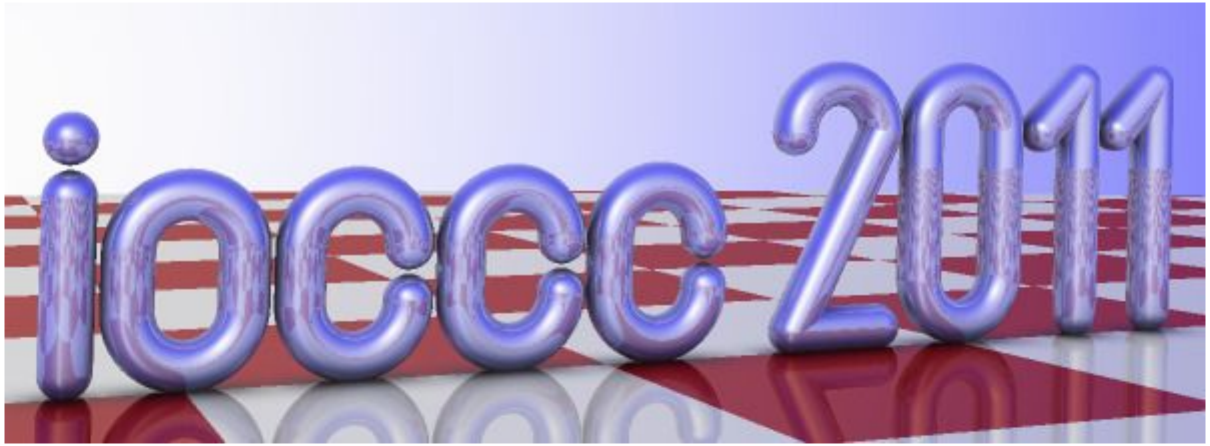
https://github.com/jensengrey/miniray

Render First Image

```
$ perf stat ./miniray_4.6 > test2.ppm

 Performance counter stats for './miniray_4.6':

          17,380.65 msec task-clock                  #    0.998 CPUs utilized
          2,498     context-switches          #    0.144 K/sec
               3    cpu-migrations            #    0.000 K/sec
               52   page-faults               #    0.003 K/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     17.415531414 seconds time elapsed

     17.378465000 seconds user
     0.000000000 seconds sys
```

Render First Image

```
$ perf stat ./miniray_4.6 > miniray_46.ppm
$ convert miniray_46.ppm miniray_46.jpg
```

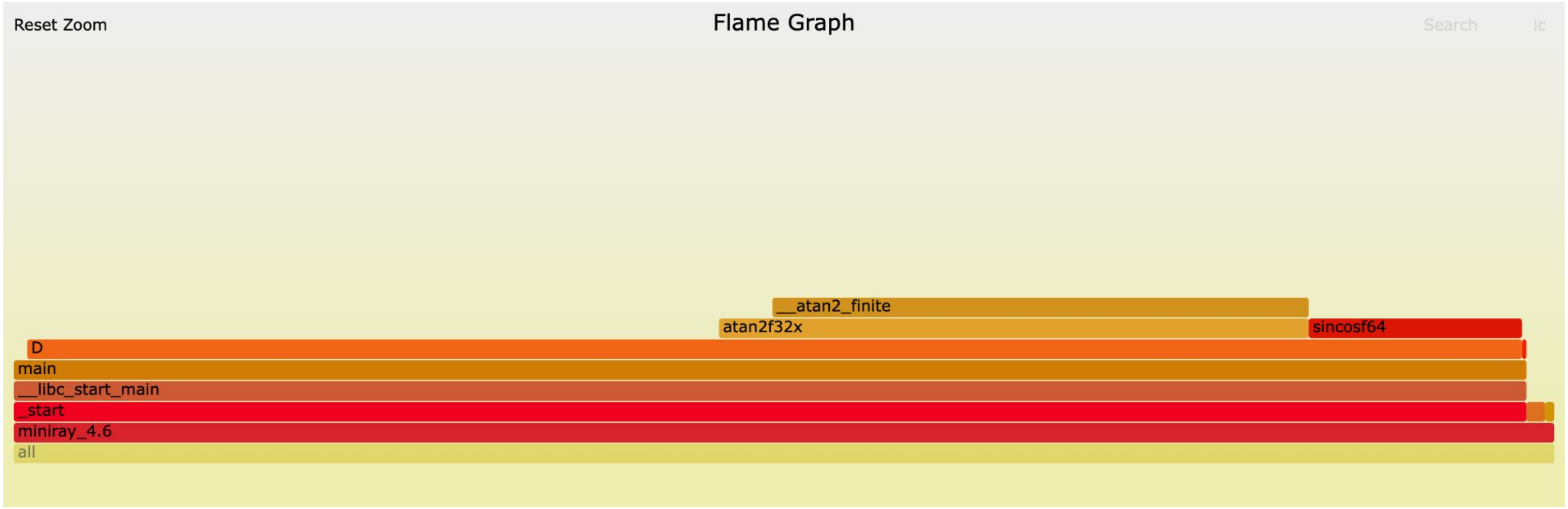


Peruse File System

```
python3 -m http.server 9999
# get your external ip
https://www.google.com/search?hl=en&q=my%20ip%20address
# add a firewall rule for your IP, allow all
gcloud compute --project=<your project> \
     firewall-rules create devaccess \
     --direction=INGRESS \
     --priority=400 \
     --network=default \
     --action=ALLOW \
     --rules=tcp \
     --source-ranges=<your ip> \
     --enable-logging
```

```
$ perf record -F 10000 -g -- ./miniray_4.6 > miniray_4.6.ppm
```

```
$ cd ~/FlameGraph;
$ perf script | ./stackcollapse-perf.pl |./flamegraph.pl > perf.svg
```



Exercise 1: Generate FlameGraph

Follow the instructions on the previous slide and generate the flamegraph.svg

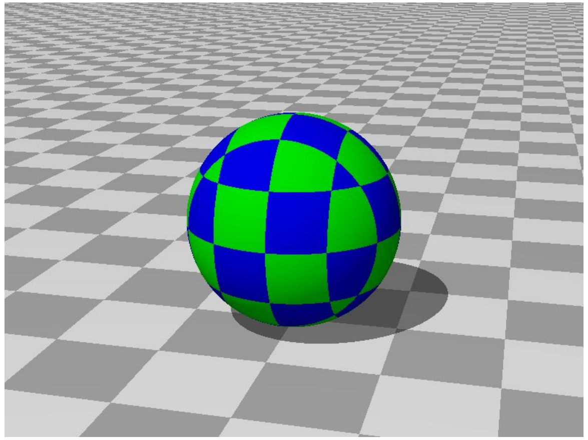
Exercise 2: What is the impact on wall clock time for including debug information?

Install Docker on Ubuntu

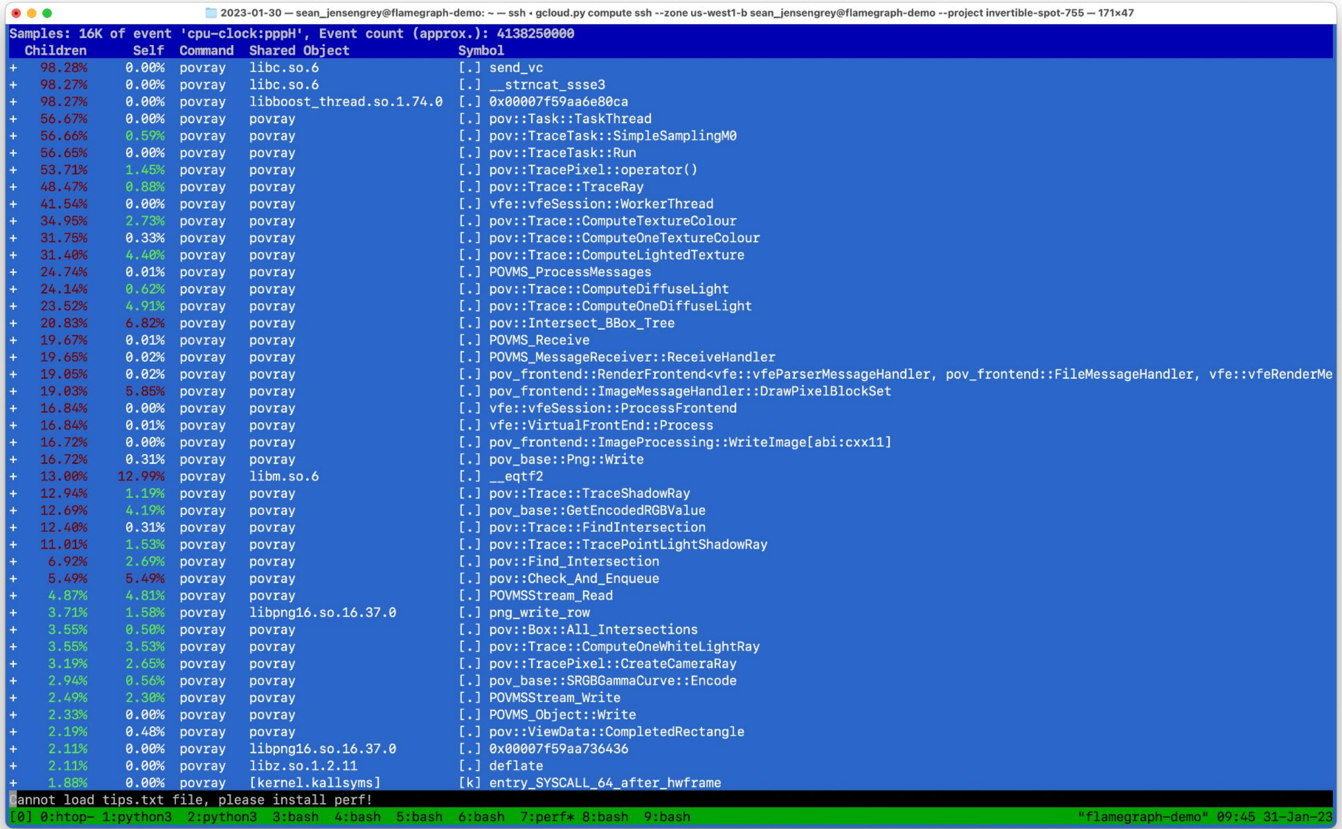
- Bring up an Ubuntu VM
- Install Docker
- Install devtools

Build Container and Render Test Image

```
git clone https://github.com/jensengrey/workshop-containers-af99
cd workshop-containers-af99/povray
sudo docker build -t povtest:1 - < povray.dockerfile
sudo docker run -t --rm -v $PWD:/povfiles --user $(id -u):$(id -g) povtest:1 povray
/povfiles/debug.pov
```



perf report --no-inline



2023-01-30 — sean_jensengrey@flamegraph-demo: ~ — ssh ‹ gcloud.py compute ssh --zone us-west1-b sean_jensengrey@flamegraph-demo --project invertible-spot-755 — 171×47

```
Samples: 16K of event 'cpu-clock:pppH', Event count (approx.): 4138250000
  Children      Self  Command  Shared Object              Symbol
+   98.28%     0.00%  povray   libc.so.6                  [.] send_vc
+   98.27%     0.00%  povray   libc.so.6                  [.] __strncat_ssse3
+   98.27%     0.00%  povray   libboost_thread.so.1.74.0  [.] 0x00007f59aa6e80ca
+   56.67%     0.00%  povray   povray                     [.] pov::Task::TaskThread
+   56.66%     0.59%  povray   povray                     [.] pov::TraceTask::SimpleSamplingM0
+   56.65%     0.00%  povray   povray                     [.] pov::TraceTask::Run
+   53.71%     1.45%  povray   povray                     [.] pov::TracePixel::operator()
+   48.47%     0.88%  povray   povray                     [.] pov::Trace::TraceRay
+   41.54%     0.00%  povray   povray                     [.] vfe::vfeSession::WorkerThread
+   34.95%     2.73%  povray   povray                     [.] pov::Trace::ComputeTextureColour
+   31.75%     0.33%  povray   povray                     [.] pov::Trace::ComputeOneTextureColour
+   31.40%     4.40%  povray   povray                     [.] pov::Trace::ComputeLightedTexture
+   24.74%     0.01%  povray   povray                     [.] POVMS_ProcessMessages
+   24.14%     0.62%  povray   povray                     [.] pov::Trace::ComputeDiffuseLight
+   23.52%     4.91%  povray   povray                     [.] pov::Trace::ComputeOneDiffuseLight
+   20.83%     6.82%  povray   povray                     [.] pov::Intersect_BBox_Tree
+   19.67%     0.01%  povray   povray                     [.] POVMS_Receive
+   19.65%     0.02%  povray   povray                     [.] POVMS_MessageReceiver::ReceiveHandler
+   19.05%     0.02%  povray   povray                     [.] pov_frontend::RenderFrontend<vfe::vfeParserMessageHandler, pov_frontend::FileMessageHandler, vfe::vfeRenderMe
+   19.03%     5.85%  povray   povray                     [.] pov_frontend::ImageMessageHandler::DrawPixelBlockSet
+   16.84%     0.00%  povray   povray                     [.] vfe::vfeSession::ProcessFrontend
+   16.84%     0.01%  povray   povray                     [.] vfe::VirtualFrontEnd::Process
+   16.72%     0.00%  povray   povray                     [.] pov_frontend::ImageProcessing::WriteImage[abi:cxx11]
+   16.72%     0.31%  povray   povray                     [.] pov_base::Png::Write
+   13.00%    12.99%  povray   libm.so.6                  [.] __eqtf2
+   12.94%     1.19%  povray   povray                     [.] pov::Trace::TraceShadowRay
+   12.69%     4.19%  povray   povray                     [.] pov_base::GetEncodedRGBValue
+   12.40%     0.31%  povray   povray                     [.] pov::Trace::FindIntersection
+   11.01%     1.53%  povray   povray                     [.] pov::Trace::TracePointLightShadowRay
+    6.92%     2.69%  povray   povray                     [.] pov::Find_Intersection
+    5.49%     5.49%  povray   povray                     [.] pov::Check_And_Enqueue
+    4.87%     4.81%  povray   povray                     [.] POVMSStream_Read
+    3.71%     1.58%  povray   libpng16.so.16.37.0        [.] png_write_row
+    3.55%     0.50%  povray   povray                     [.] pov::Box::All_Intersections
+    3.55%     3.53%  povray   povray                     [.] pov::Trace::ComputeOneWhiteLightRay
+    3.19%     2.65%  povray   povray                     [.] pov::TracePixel::CreateCameraRay
+    2.94%     0.56%  povray   povray                     [.] pov_base::SRGBGammaCurve::Encode
+    2.49%     2.30%  povray   povray                     [.] POVMSStream_Write
+    2.33%     0.00%  povray   povray                     [.] POVMS_Object::Write
+    2.19%     0.48%  povray   povray                     [.] pov::ViewData::CompletedRectangle
+    2.11%     0.00%  povray   libpng16.so.16.37.0        [.] 0x00007f59aa736436
+    2.11%     0.00%  povray   libz.so.1.2.11             [.] deflate
+    1.88%     0.00%  povray   [kernel.kallsyms]          [k] entry_SYSCALL_64_after_hwframe
Cannot load tips.txt file, please install perf!
[0] 0:htop- 1:python3  2:python3  3:bash  4:bash  5:bash  6:bash  7:perf* 8:bash  9:bash          "flamegraph-demo" 09:45 31-Jan-23
```

Hacking Dockerfiles

- Don't suffer in docker build loop
- Spin up a VM with the same base image as your container, debug and then port into Dockerfile

Single Best Docker+Perf Tutorial

- https://gendignoux.com/blog/2019/11/09/profiling-rust-docker-perf.html

Docker Locks Down Specific Syscalls

- https://docs.docker.com/engine/security/seccomp/

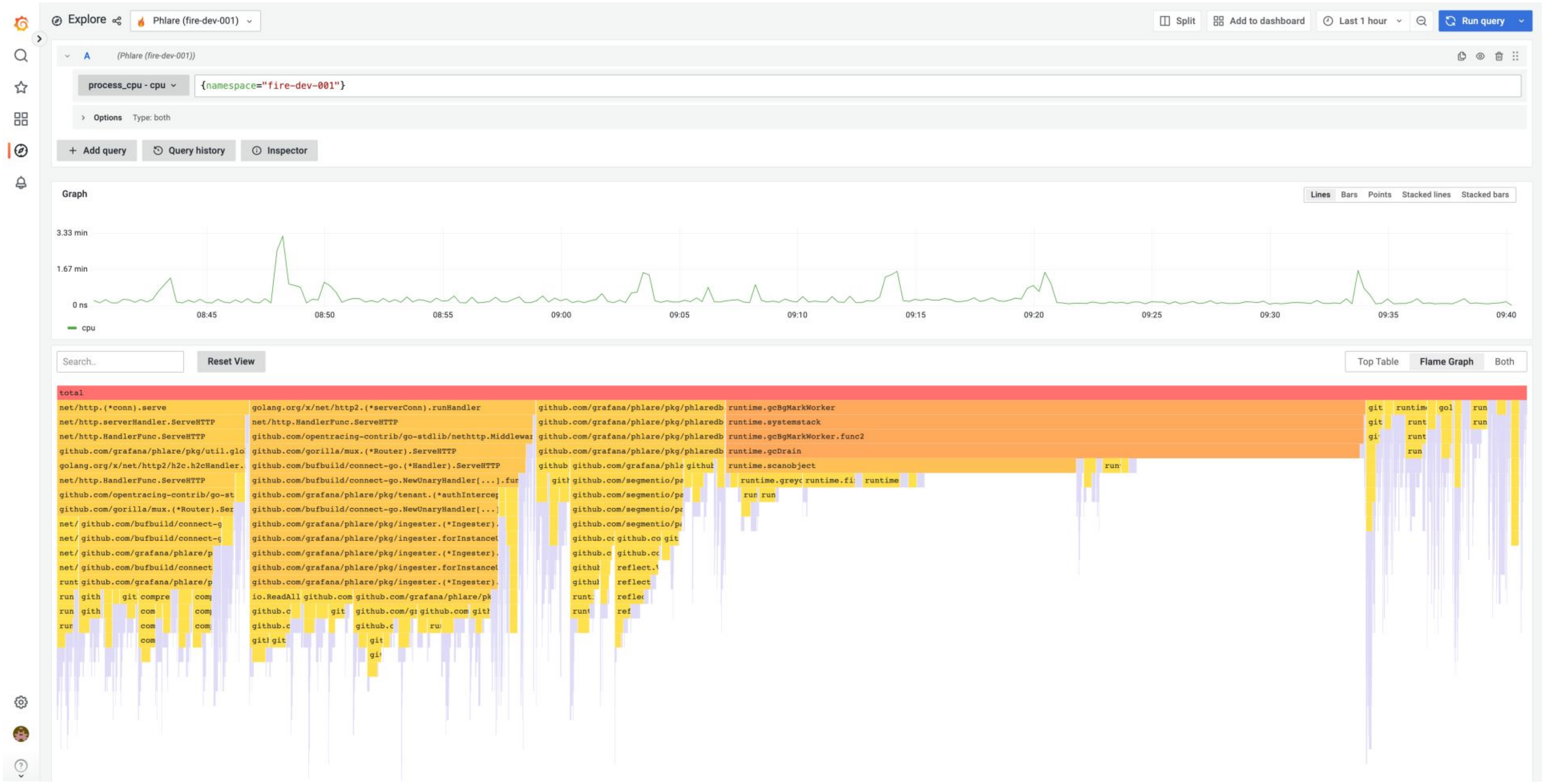
Modify Build Command to Support Perf

```
cd povray/unix && \
     ./prebuild.sh && \
     cd .. && \
     CXXFLAGS="-fno-omit-frame-pointer" ./configure --enable-debug --enable-profile
COMPILED_BY="povray-perf-builder" && \
     make -j4 && \
     make install
```

Generate Perf Trace

```
sudo docker run -t -v $PWD:/povfiles --security-opt seccomp=unconfined povtest-perf:1 perf
record -F 100 -a -g -- povray /povfiles/debug.pov;
```

Continuous Profiling in Grafana Phlare



```
{namespace="fire-dev-001"}
```

process_cpu - cpu

Options Type: both

Explore
Phlare (fire-dev-001)
Split
Add to dashboard
Last 1 hour
Run query
A (Phlare (fire-dev-001))
Add query
Query history
Inspector
Graph
Lines
Bars
Points
Stacked lines
Stacked bars
3.33 min
1.67 min
0 ns
08:45
08:50
08:55
09:00
09:05
09:10
09:15
09:20
09:25
09:30
09:35
09:40
cpu
Search..
Reset View
Top Table
Flame Graph
Both
total
net/http.(*conn).serve
net/http.serverHandler.ServeHTTP
net/http.HandlerFunc.ServeHTTP
golang.org/x/net/http2.(*serverConn).runHandler
net/http.HandlerFunc.ServeHTTP
github.com/opentracing-contrib/go-stdlib/nethttp.Middlewar
github.com/gorilla/mux.(*Router).ServeHTTP
github.com/bufbuild/connect-go.(*Handler).ServeHTTP
github.com/bufbuild/connect-go.NewUnaryHandler[...].fur
github.com/grafana/phlare/pkg/tenant.(*authIntercep
github.com/grafana/phlare/pkg/ingester.(*Ingester).
github.com/grafana/phlare/pkg/ingester.forInstance
io.ReadAll
github.com/grafana/phlare/pkg/phlaredb
github.com/segmentio/pa
reflect.
runtime.gcBgMarkWorker
runtime.systemstack
runtime.gcBgMarkWorker.func2
runtime.gcDrain
runtime.scanobject
runtime.greyc
runtime.fi
runtime

Grafana Phlare

- https://grafana.com/oss/phlare/
- https://github.com/grafana/phlare
- https://grafana.com/docs/phlare/latest/
- https://grafana.com/docs/phlare/latest/operators-guide/deploy-kubernetes/
- https://grafana.com/tutorials/
- https://www.youtube.com/watch?v=ORINvZAurIY