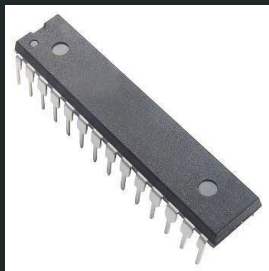


とにかく簡単にビルドの プロファイルを取りたい!

@Kernel/VM探検隊@関西 11回目



| | |
|-----------|---|
| Name | Akira Kawata |
| Biography | https://akawashiro.com/ |
| twitter | @a_kawashiro |
| bluesky | akawashiro.bsky.social |

デモ

<https://akawashiro.com/software/s-traceprof/>



ソースコード

<https://github.com/akawashiro/straceprof/tree/main/straceprof-js>



自己紹介

- 河田 旺 (Akira Kawata) / a_kawashiro
- 仕事
 - 機械学習専用 ASIC 向けのソフトウェア作成
- 非仕事
 - Binary Hacks Rebooted
 - 初見のソフトウェアを読むときは
とりあえずビルドして print を挟んでみる



O'REILLY
オライリー・ジャパン

Binary Hacks Rebooted

低レイヤの世界を探検するテクニック89選



河田 旺、小池 悠生、渡邉 慶一 著
佐伯 学哉、荒田 実樹
鈴木 創、中村 孝史、竹腰 剛
光成 滋生、hikalium、浜地 慎一郎 寄稿

ビルドとは何か？

ソフトウェアのビルド（英: build）は、プログラミング言語で書かれたソースコードファイルや各種リソースファイルを独立したソフトウェア生成物に変換するコンピュータ上で実行されるプロセス、またはその結果を指す。ビルドの最終生成物はバイナリ形式の実行ファイルであったり、再利用可能なライブラリであったり、バイトコードあるいはそれらをまとめたアーカイブであったりすることもある。[1]

[1]: [ビルド \(ソフトウェア\) - Wikipedia](#)

ビルドの例

| | ビルドシステム | 所要時間 |
|--------------|----------------|-----------|
| Linux kernel | make | 3分ぐらい |
| glibc | make | 3分ぐらい |
| Julia | make | 9分ぐらい |
| PyTorch [1] | Python + CMake | 14分ぐらい |
| コンテナイメージ | Dockerfile(?) | 1分 から 1時間 |

[1]: GPU 関連のバックエンドをすべて無効化したコンフィグです

ビルドが遅い

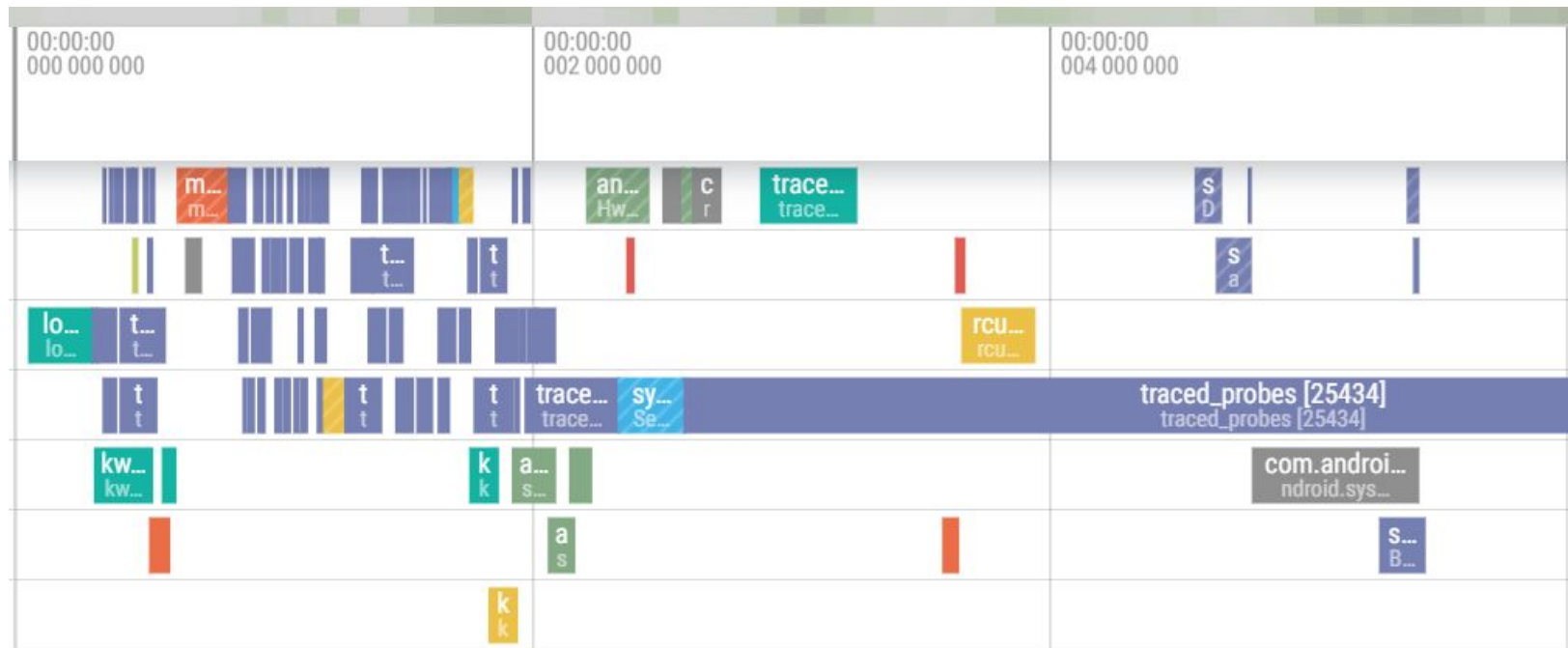
- ビルドが遅いとつらい
 - 変更してから効果を確認するまでの時間が伸びる
 - CI の時間が伸びる
- ビルドは様々な理由で遅くなる
 - 外部からのパッケージの取得
 - 不要なターゲットのビルド
 - ソースコードのコンパイル

⇒ 高速化したい...

とにかく簡単にビルドのプロファイルを取りたい!

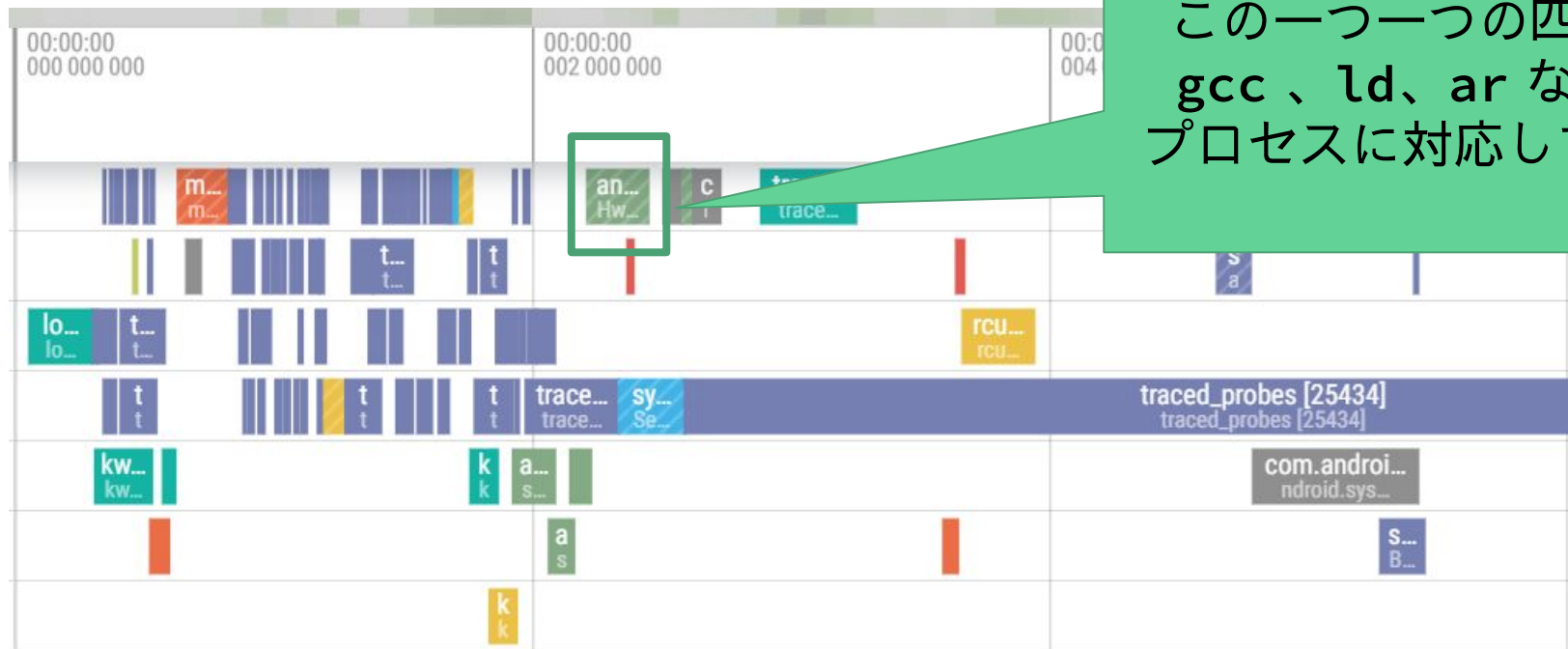
欲しいプロファイラのイメージ

<https://ui.perfetto.dev/> のビルド版



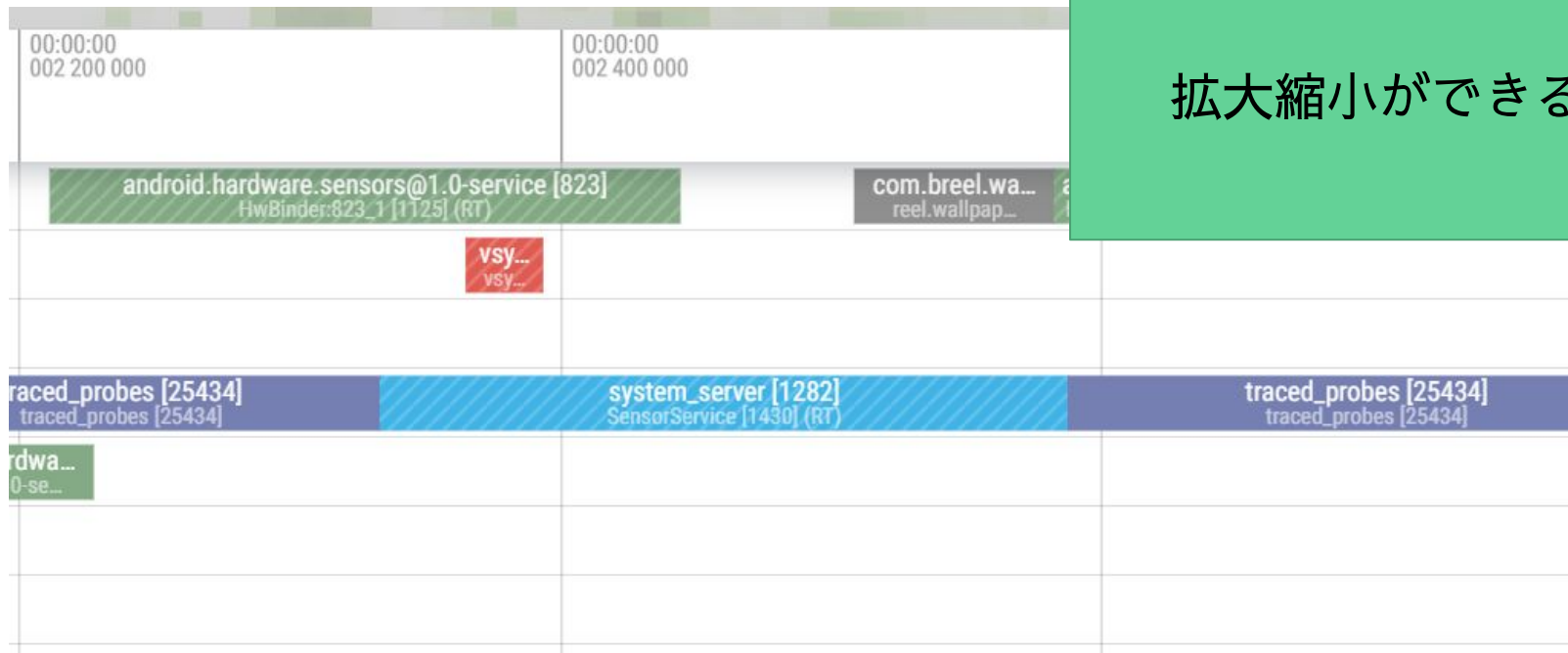
欲しいプロファイラのイメージ

<https://ui.perfetto.dev/> のビルド版



欲しいプロファイラのイメージ

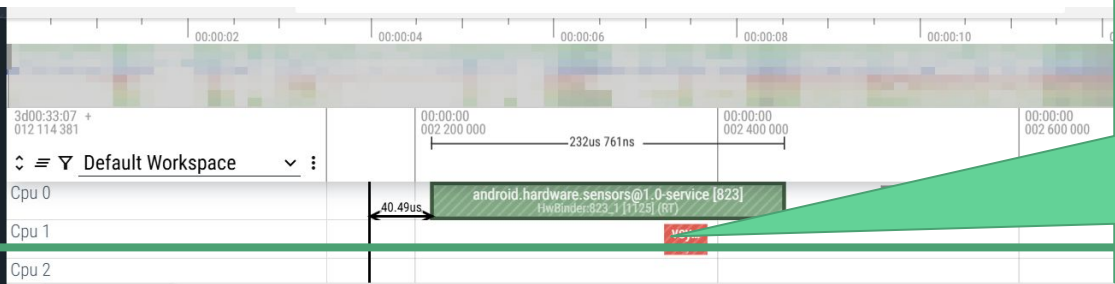
<https://ui.perfetto.dev/> のビルド版



欲しいプロファイラのイメージ

<https://ui.perfetto.dev/> のビルド版

イベントをクリックすると
詳細な情報が出てくる



CPU Sched Slice /vendor/bin/hw/android.hardware.sensors@1.0-service [823]

Details

Process /vendor/bin/hw/android.hardware.sensors@1.0-service [823]
Thread [HwBinder:823_1 \[1125\]](#)
Cmdline /vendor/bin/hw/android.hardware.sensors@1.0-service
Start time [00:00:00.002 211 458](#)
Duration [232us 761ns](#)
Priority 89 (real-time)
End State Sleeping
SQL ID [sched\[153\]](#)
Thread HwBinder:823_1 1125
Process /vendor/bin/hw/android.hardware.sensors@1.0-service 823

Scheduling Latency

Wakeup @ [00:00:00.002 170 885](#) on CPU 4 by
P: /vendor/bin/hw/android.hardware.sensors@1.0-service [823]
T: sensors@1.0-ser [1123]

←→ Scheduling latency: [40us 573ns](#)
This is the interval from when the task became eligible to run (e.g. because of notifying a wait queue it was suspended on) to when it started running.

既存のプロファイラと問題点

- 世の中には様々なプロファイラがあるが、
どれもビルドをプロファイルするのに使いにくい

既存のプロファイラと問題点—gprof

- 使い方
 - 実行ファイルを gcc でコンパイルする際に
-pg オプションをつける
 - 実行すると gmon.out ファイルが出てくる
 - gprof <実行ファイル> gmon.out を実行する
- 不満
 - 単一のプログラムをプロファイルするためのもの
 - ビルド全体をプロファイルするのには使えない

既存のプロファイラと問題点—perfetto

- 使い方
 - linux-tracing にマルチプロセスでの使い方が書いてある
 - 専用の tmux セッションの中でコマンドを実行する
- 問題点
 - プロファイラを利用する手順が複雑
 - CI のような制限された状況下で動かすのが難しい
 - デフォルトでは関数単位の情報記録され、大きくなりすぎる
 - ほしいのはプロセス単位の情報

既存のプロファイルと問題点—ninjatracng

- 使い方
 - CMake などでビルドする際にバックエンドとしてninjaを選択する
 - ビルドを実行すると ninja.log というファイルが出てくる
 - ninjatracng で ninja.log を処理すると chrome で可視化できる trace.json が出てくる
- 問題点
 - ビルド対象が ninja を使っていない場合使えない
 - make、Dockerfile、シェルスクリプト等のプロファイルをとりたい

理想のプロファイラの要件

- すぐにインストールできる
- どんなビルドシステムを利用しても使える
- ビルド中に起動した各プロセスの開始時間と終了時間が可視化できる

strace

- Linux で動くデバッグツール
 - どのディストリビューションでもすぐインストールできる
- プロセスが発行したシステムコールを監視できる
 - **--follow-forks** オプションをつけると
子プロセスや孫プロセスも監視できる

[再掲] 理想のプロファイラの要件

- すぐにインストールできる
- どんなビルドシステムを利用していても使える
- ビルド中に起動した各プロセスの開始時間と終了時間が可視化できる

strace ができるのでは？

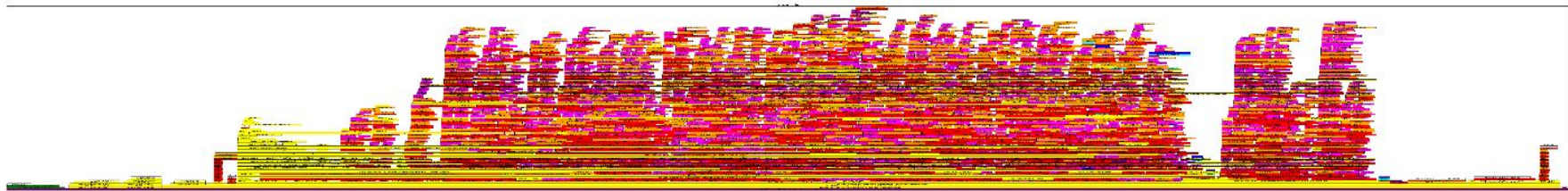
- すぐにインストールできる
 - OK
- どんなビルドシステムを利用しても使える
 - OK
- ビルド中に起動した各プロセスの開始時間と終了時間が可視化できる
 - `execve`、`execveat` と `exit`、`exit_group` システムコールを監視すると大体のプロセスは補足できる
 - このビジュアライザ部分だけ作ればプロファイラになる

ビジュアライザを作りました

- 使い方
 - 1. strace をつけてログをとる
 - 2. ログをビジュアライザに入れて画像を作る

Python バージョン

1. `$ strace --trace=execve,execveat,exit,exit_group \`
 `-f -ttt --string-limit=1000 --output=straceprof.log \`
 `--seccomp-bpf <command to profile>`
2. `$ straceprof --log=straceprof.log --output=straceprof.png`

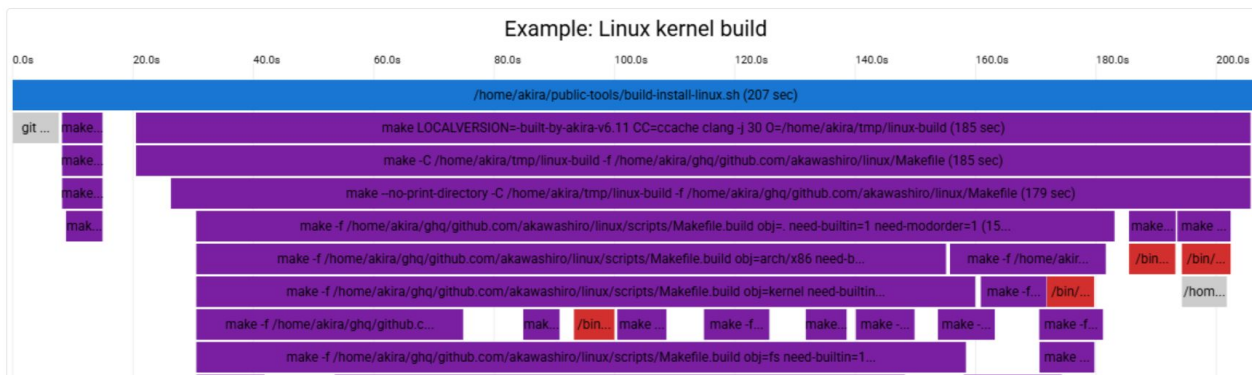


Python バージョンの問題点

- matplotlib で画像を作っているのでUIの細部が詰められない
 - 画像サイズを小さくすると文字がつぶれる
 - マウスホバーが使えない
- 「簡単」ではない
 - 可視化するのに何かをインストールするのは面倒
 - ブラウザで動いてほしい

Web バージョン

1. `$ strace --trace=execve,execveat,exit,exit_group \`
`-f -ttt --string-limit=1000 --output=straceprof.log \`
`--seccomp-bpf <command to profile>`
2. <https://akawashiro.com/software/straceprof/>
にログをアップロード



デモ

straceprof

straceprof is a profiler designed for multi-process programs. straceprof can take profile of any process when you can run it under strace. It is particularly well-suited for profiling build processes such as those initiated by make, cmake, shell scripts, or docker build. Upload the result of the following command and visualize it.

strace -trace=execve,execveat,exit,exit_group --follow-forks --string-limit=1000 -ttt --output=straceprof.log <command to profile>

COPY

UPLOAD LOG

Load a sample Log

Ruby build

Threshold to show processes (sec)

0 sec

0.0 sec

95.5

Time range to visualize (sec)

0.0 sec

95.5

Filter processes by regexp

^.*\$

Default: ^.*\$ (matches all processes)



終わり

FAQ:

- Q: strace 自体のオーバーヘッドは？
- A: ruby のビルドで37秒 vs 38秒ぐらい
 - --seccomp-bpf オプションが必要
- Q: どれくらいのプロセスを取りこぼしているの？
- A: console.log に `Parsed 5700 execve, 0 execveat, 49 exit, 3993 exit_group` のような行が出ています。これは Ruby の例です。このあたりは調査中です。