Get started with Performance Advisor

8.8

Tutorial

Non-Confidential

Issue 00

102478_0808_00_en

Get started with Performance Advisor

Tutorial

Release information

Document history

Issue	Date	Confidentiality	Change
0101-00	27 June 2022	Non-Confidential	LWI directory update
0102-00	29 March 2023	Non-Confidential	Updated for version 8.4, added troubleshooting, example report and general improvements
0103-00	21 April 2023	Non-Confidential	Updated for version 8.5, Performance Advisor is now part of Streamline. Install directory and cli changes.
0806-00	15 June 2023	Non-Confidential	Updated for version 8.6. lwi_me.py script renamed to streamline_me.py and change to –lwi-mode default setting. Vulkan 1.3 supported.
0808-00	5 October 2023	Non-Confidential	Updated for version 8.8. Simple Streamline connection mode for Android. The appropriate GPU template is now automatically applied.

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at https://www.arm.com/company/policies/trademarks.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

To get quick analytics on how your Android application performs on a connected mobile device, use Performance Advisor to generate an easy-to-read performance summary from a Streamline profile.

This tutorial describes how to:

- Perform Setup tasks to prepare your computer and device.
- Run the streamline_me.py script to enable Streamline to collect frame data from the device. You can only generate a Performance Advisor report from a Streamline profile that was captured by first running this script.
- Capture a Streamline profile. Once this is done, you will be able to run the Performance Advisor command on the saved file to generate a performance report.
- Generate a performance report in HTML format. There is a walkthrough of an Example Performance Advisor report that describes how to interpret the data collected.
- Generate a JSON report so that you can feed data into other systems to monitor perofrmance over time.
- Access documentation for all the Command-line options available for the `streamline_me.py` script and `streamline-cli -pa` command.

Note

This tutorial applies to the latest released version of Arm Mobile Studio. In version 2023.1, Performance Advisor became part of the Streamline tool, resulting in a change to the install location of the Performance Advisor files, and to the command used to generate reports. If you are using a version earlier than 2023.1, follow the instructions in the previous version of this tutorial instead.

The following training video introduces Performance Advisor, and demonstrates how to capture a Streamline profile, then generate and analyze a Performance Advisor report: Android profiling with Performance Advisor.

2. Setup tasks

Follow these steps to set up your computer and device so that you can analyze your application with Streamline and Performance Advisor.

Before you begin

- Performance Advisor supports applications built with OpenGL ES versions 2.0 to 3.2, or Vulkan versions 1.0 to 1.3. For OpenGL ES applications, your device must be running Android 10 or later. For Vulkan applications, your device must be running Android 9 or later.
- Ensure you have Android Debug Bridge (ADB) installed. ADB is available with the Android SDK platform tools, which are installed as part of Android Studio, or you can download them separately here.
- Performance Advisor uses a Python script to connect to your device. To run this script, you will need Python 3.5 or later installed.

Procedure

1. Download Arm Mobile Studio and follow the installation instructions in the Arm Mobile Studio Release Note.
2. Edit your `PATH` environment variable so you can run Streamline's `Streamline-cli -pa` command from any directory. This step is not necessary on Windows, as this is done automatically when Arm Mobile Studio is installed.
   - On macOS, we provide a launcher file that opens the Terminal application, and sets your `PATH` environment variable. Navigate to the `<installation_directory>/streamline` directory, and double-click the `Streamline-cli-launcher` file. Your computer will ask you to allow Streamline to control the Terminal application. Confirm this.

     Alternatively, you can edit the `/etc/paths` file to add the path manually. The path is: `/<installation_directory>/streamline`
   - On Linux, edit your `PATH` environment variable to add the path to the Performance Advisor executable. Add this command to the `.bashrc` file in your home directory, so that this environment variable is set whenever you initialize a shell session;

     ```
     PATH=$PATH:/<installation_directory>/streamline
     ```

   You can also add the paths to the location of the ADB and Python executables, which makes it possible to run `adb` and `python3` commands from any directory.
3. Connect your device to your computer through USB. Ensure that your device is set to Developer mode.
4. On your device, go to Settings > Developer Options and enable USB Debugging. If your device asks you to authorize connection to your computer, confirm this.

You can test the connection by entering the `adb devices` command in a command terminal. If successful, the command returns the device ID.

```
adb devices
List of devices attached
ce12345abcdf1a1234        device
```

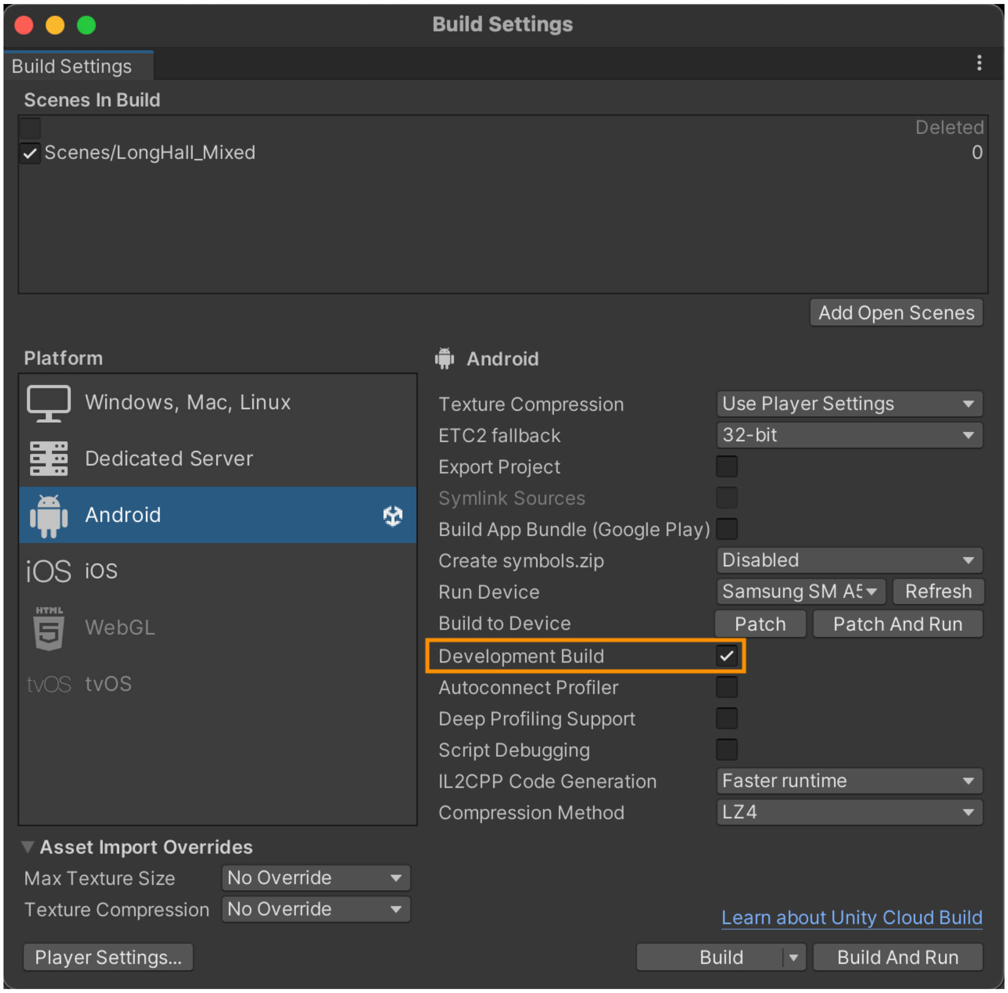
If you see that the device is listed as unauthorized, try disabling and re-enabling USB debugging on the device, and accept the authorization prompt to enable connection to the computer.

5. Install a debuggable build of the application you want to profile on the device.

   To do this in Android Studio, create a build variant that includes `debuggable true` in the build configuration. Or you can set `android:debuggable=true` in the application manifest file.

   To do this in Unity, select Development Build under File > Build Settings when building your application.

Figure 2-1: Unity Build Settings



☑ Scenes/LongHall_Mixed 0

Texture Compression: Use Player Settings
ETC2 fallback: 32-bit
Export Project ☐
Symlink Sources ☐
Build App Bundle (Google Play) ☐
Create symbols.zip: Disabled
Run Device: Samsung SM A5
Build to Device
Development Build ☑
Autoconnect Profiler ☐
Deep Profiling Support ☐
Script Debugging ☐
IL2CPP Code Generation: Faster runtime
Compression Method: LZ4

Asset Import Overrides
Max Texture Size: No Override
Texture Compression: No Override

Next steps

Now that your computer and device are connected and set up, the next step is to Run the streamline_me.py script, which enables Streamline to capture the extra frame data from the device that Performance Advisor needs.

3. Run the streamline_me.py script

Performance Advisor runs on a Streamline capture file, so the first step is to take a capture with Streamline. Streamline must capture extra frame data from the device, which Performance Advisor needs to generate a report. To capture the extra frame data, you must first run the provided Python script, `streamline_me.py`.

About this task

The `streamline_me.py` script is located in the Arm Mobile Studio installation directory:

```
<installation_directory>/streamline/bin/android
```

This script does the following:

- Temporarily installs a daemon application on your device, called `gatord`, which Streamline uses to collect counter data.
- Temporarily installs the OpenGL ES or Vulkan layer library file on your device, which is needed to collect frame data.
- Enables you to specify options for the capture, such as whether to capture screenshots when the FPS drops below a certain threshold.

Procedure

1. In a terminal window, navigate to the following folder in your Arm Mobile Studio installation directory, where the Python script `streamline_me.py` is located:

```
cd <installation_directory>/streamline/bin/android
```

Note

Do not move the script to another location, as it needs to be able to locate other files in your installation directory.

2. To run the script, type:

```
python3 streamline_me.py --lwi-mode=counters [options]
```

Note

The `--lwi-mode=counters` option enables frame boundary instrumentation. From Arm Mobile Studio version 2023.2, this option is required unless you have manually included frame boundary annotations in your application. For more information about using manual annotations, refer to the Performance Advisor user guide

You can use a range of other options to control the capture. For example, if you want to capture screenshots from the application when the FPS drops below a certain threshold:

a. Include the option `--lwi-mode=screenshots`

b. Specify the FPS threshold with `--lwi-fps-threshold=<val>`.

c. Specify an output directory for the images with `--lwi-out-dir=<location>`. When you Generate a performance report, you will need to specify this location.

For example:

```
python3 streamline_me.py --lwi-mode=screenshots --lwi-fps-threshold=50 --lwi-out-
dir=$HOME/Documents/MyGameProfile/SlowFrames
```

Note

By default, the `streamline_me.py` script captures the OpenGL ES API. To capture the Vulkan API, use the `--lwi-api=vulkan` option.

For a full list of the available options, you can use the `python3 streamline_me.py -h` command, or refer to the Performance Advisor user guide.

3. The script returns a numbered list of the Android package names for the debuggable applications that are installed on your device.

```
Searching for devices:
 RZ8MC03VVEW / SM-A505FN found

Select a device:
 Auto-selected RZ8MC03VVEW / SM-A505FN

Searching for debuggable packages:
    5 debuggable packages found

Select a debuggable packages:
     1) com.Arm.DarkArms
     2) com.UnityTechnologies.BoatAttack
     3) com.arm.malideveloper.openglessdk.occlusionculling
     4) com.arm.pa.paretrace
     5) com.sample.texturedteapot
     0) Exit script
```

4. Enter the number of the application that you want to profile.

   The script identifies the GPU in the device, installs the daemon application and layer library, then waits for you to complete the capture in Streamline.

```
 Select entry: 4

 Selected com.arm.pa.paretrace

Searching for a Mali GPU:
 Mali-G72 GPU found
--lwi-gles-layer-lib-path not specified, assuming /Applications/
Arm_Mobile_Studio_2023.2/streamline/bin/android/arm64/libGLESLayerLWI.so

Installing OpenGL ES debug layer

1) Configure and profile using Streamline
```

```
2) Press <Enter> here after capture has completed to finish.
```

Warning

Leave the terminal window open, as you need to come back to it after the capture is complete, to stop the script. When the script ends, any captured screenshots are saved to the directory you specified, and the daemon application and layer library are uninstalled from the device. Do not unplug the device until the script has ended.

Next steps

Now that the device is ready to collect data, the next step is to Capture a Streamline profile.

4. Capture a Streamline profile

Take a capture with Streamline, to collect a performance profile of your application running on the connected device.

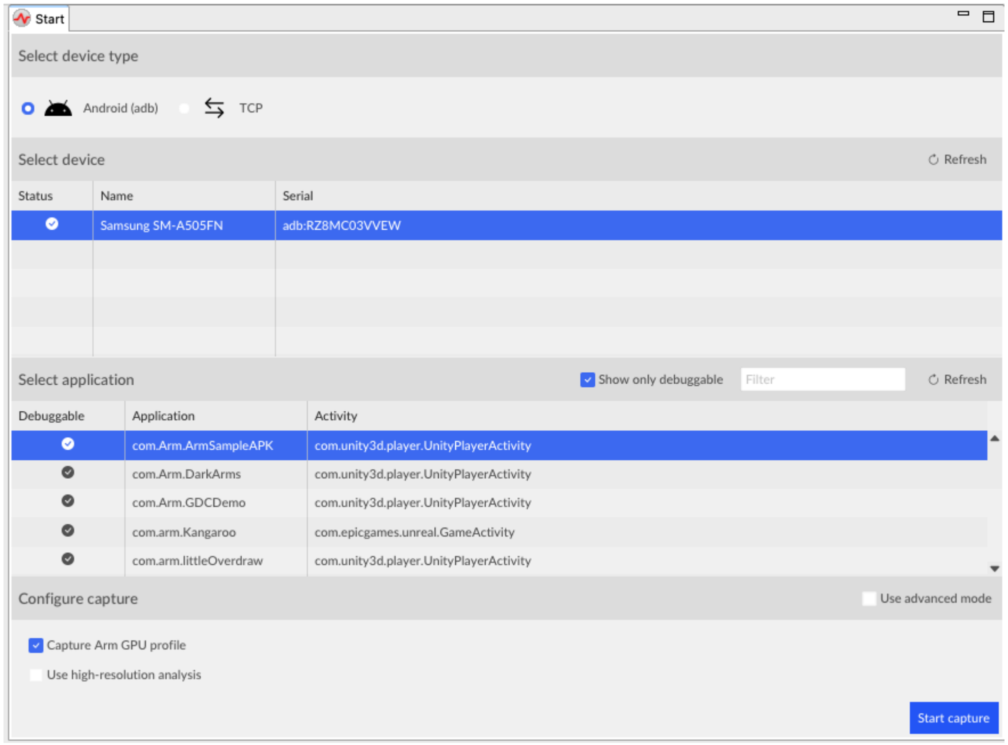
Procedure

1. Launch Streamline:
   a. On Windows, from the Start menu, navigate to Arm MS <version> and select Arm MS Streamline <version>.
   b. On macOS, go to the `<install_directory>/streamline` folder, and double-click the `Streamline.app` file.
   c. On Linux, navigate to the `<install_directory>/streamline` directory in a terminal, and run the Streamline file:

```
cd <install_directory>/streamline
./Streamline
```

2. Use the Start tab in Streamline to select your device, and the application you want to profile.

Figure 4-1: Connect to your device in Streamline



● Android (adb) ○ TCP

Status	Name	Serial
✓	Samsung SM-A505FN	adb:RZ8MC03VVEW

☑ Show only debuggable

Debuggable	Application	Activity
✓	com.Arm.ArmSampleAPK	com.unity3d.player.UnityPlayerActivity
✓	com.Arm.DarkArms	com.unity3d.player.UnityPlayerActivity
✓	com.Arm.GDCDemo	com.unity3d.player.UnityPlayerActivity
✓	com.arm.Kangaroo	com.epicgames.unreal.GameActivity
✓	com.arm.littleOverdraw	com.unity3d.player.UnityPlayerActivity

☐ Use advanced mode

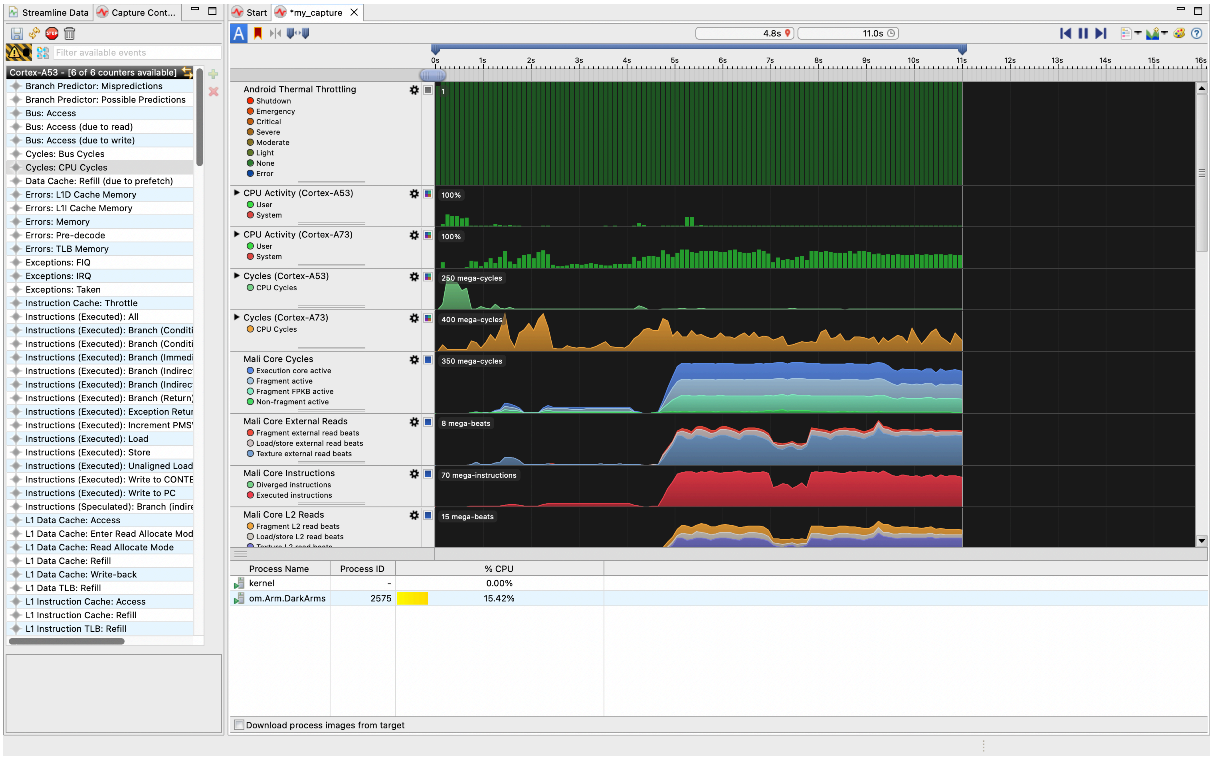
☑ Capture Arm GPU profile

☐ Use high-resolution analysis

With Capture Arm GPU profile enabled, Streamline will select a counter template appropriate for the GPU in your device. If you would rather build your own counter configuration, select Use advanced mode in the Configure capture section, and choose Select counters.

3. Select Start Capture and specify the name and location of the capture file that Streamline will create when the capture completes. Streamline then switches to Live view and the application starts on the device.
4. Perform your test scenario on the device. Streamline shows the performance counter activity being collected.

Figure 4-2: Streamline Live view


Streamline Data
Capture Cont...
Filter available events
Cortex-A53 - [6 of 6 counters available]
Branch Predictor: Mispredictions
Branch Predictor: Possible Predictions
Bus: Access
Bus: Access (due to read)
Bus: Access (due to write)
Cycles: Bus Cycles
Cycles: CPU Cycles
Data Cache: Refill (due to prefetch)
Errors: L1D Cache Memory
Errors: L1I Cache Memory
Errors: Memory
Errors: Pre-decode
Errors: TLB Memory
Exceptions: FIQ
Exceptions: IRQ
Exceptions: Taken
Instruction Cache: Throttle
Instructions (Executed): All
Instructions (Executed): Branch (Conditi
Instructions (Executed): Branch (Conditi
Instructions (Executed): Branch (Immedi
Instructions (Executed): Branch (Indirec
Instructions (Executed): Branch (Indirec
Instructions (Executed): Branch (Return)
Instructions (Executed): Exception Retur
Instructions (Executed): Increment PMS
Instructions (Executed): Load
Instructions (Executed): Store
Instructions (Executed): Unaligned Load
Instructions (Executed): Write to CONTE
Instructions (Executed): Write to PC
Instructions (Speculated): Branch (indire
L1 Data Cache: Access
L1 Data Cache: Enter Read Allocate Mod
L1 Data Cache: Read Allocate Mode
L1 Data Cache: Refill
L1 Data Cache: Write-back
L1 Data TLB: Refill
L1 Instruction Cache: Access
L1 Instruction Cache: Refill
L1 Instruction TLB: Refill
Start
*my_capture
A
4.8s
11.0s
0s
1s
2s
3s
4s
5s
6s
7s
8s
9s
10s
11s
12s
13s
14s
15s
16s
Android Thermal Throttling
Shutdown
Emergency
Critical
Severe
Moderate
Light
None
Error
1
CPU Activity (Cortex-A53)
User
System
100%
CPU Activity (Cortex-A73)
User
System
100%
Cycles (Cortex-A53)
CPU Cycles
250 mega-cycles
Cycles (Cortex-A73)
CPU Cycles
400 mega-cycles
Mali Core Cycles
Execution core active
Fragment active
Fragment FPKB active
Non-fragment active
350 mega-cycles
Mali Core External Reads
Fragment external read beats
Load/store external read beats
Texture external read beats
8 mega-beats
Mali Core Instructions
Diverged instructions
Executed instructions
70 mega-instructions
Mali Core L2 Reads
Fragment L2 read beats
Load/store L2 read beats
Texture L2 read beats
15 mega-beats
Download process images from target

Process Name	Process ID	% CPU
kernel	-	0.00%
om.Arm.DarkArms	2575	15.42%

5. Click Stop to end the capture. Streamline finishes the capture and stores the capture file in the location you specified.
6. IMPORTANT: Switch back to the terminal running the `streamline_me.py` script and press any key to stop it. The script stops all processes that it started and removes the daemon application from the device.

   Any slow frame screenshots that were captured are saved to the location you specified when running the script.

Next steps

Later, for a more deep dive analysis, you can investigate the Streamline capture to see in detail how the device performed while running your application. You can also refer to the Mali GPU training video, Episode 3-3 Streamline to learn how to interpret the results in Streamline.

But for now, you can close Streamline and in the next section, we will Generate a performance report with Performance Advisor.

5. Generate a performance report

In this section, we will use the `Streamline-cli -pa` command to run Performance Advisor on a Streamline capture file to generate a performance report.

Procedure

1. In a terminal, navigate to the location of your Streamline capture file.

```
cd $HOME/Documents/my_captures
ls
my_capture.apc
```

2. Run the `Streamline-cli -pa` command to generate an HTML report. By default, the report is saved to the current location.

```
Streamline-cli -pa my_capture.apc [options]
```

   You can pass a range of different options to this command to control how it runs. For example:

   a. Use the `--directory` option to specify a different location to save the report.

   b. Add metadata for the report to help you identify the run, such as the application name, device name and build name. Use the `--application-name`, `--device-name`, `--build-name` and `build-timestamp` options to do this.

   c. If you captured screenshots of the game by specifying `--lwi-mode=screenshots` when running the `streamline_me.py` script, you will need to specify the directory where those screenshots are saved, so that they will be included in the report. Use the `--frame-capture` option to do this.

```
Streamline-cli -pa --directory=$HOME/Documents/Reports --application-
name="Manhattan" --device-name="A50" --build-name="Build1" --build-
timestamp=1st March 2023 --frame-capture=$HOME/Documents/MyGameProfile/
SlowFrames my_capture.apc
```

   For a full list of the available command options, refer to The Streamline-cli -pa command in the Performance Advisor user guide.

   Tip

   To pass a list of command-line options to this command, use `Streamline-cli -pa my_capture.apc "@my_filename"`. Refer to the user guide for an example.

3. Performance Advisor processes the capture and returns the location where it has saved the HTML report file:

```
Importing capture...
Fetching data...
Preparing report type html...
Processing data...
Generating report type html...
```

```
Report "/Users/username/Documents/Reports/performance_advisor-230307-135316.html"
 successfully generated
```

Open the HTML file in a web browser to view it.

Next steps

Next, view the Example Performance Advisor report.

For more information about how to interpret the report, refer to the Mali GPU training video, Episode 3-2 Performance Advisor.

You can also Generate a JSON report with the `streamline-cli -pa` command, which can be fed into a performance dashboard to monitor changes over time.

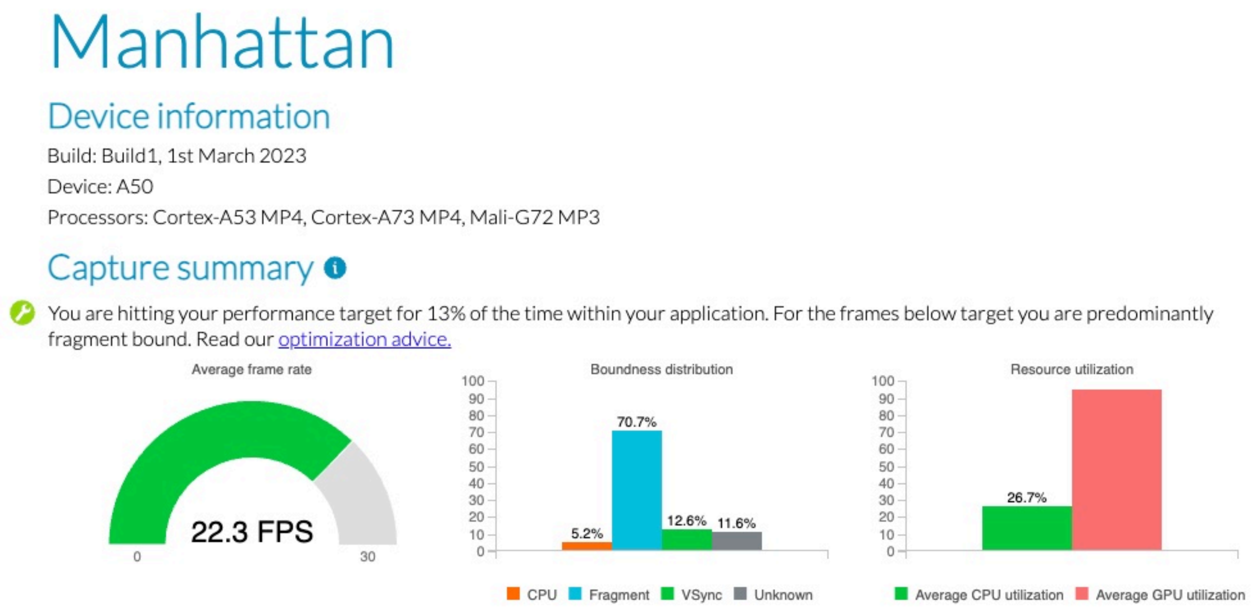
6. Example Performance Advisor report

This section presents an example Performance Advisor report, and describes the charts and data that are presented. You can also watch this video for a breakdown of a Performance Advisor report.

Report summary

The first part of the report gives details about the device and application that were tested, and provides a quick summary of performance. We can see the average FPS, a breakdown of workload, and the average utilization of the CPU and GPU in the device.

Figure 6-1: Report summary



Manhattan

Device information

Build: Build1, 1st March 2023

Device: A50

Processors: Cortex-A53 MP4, Cortex-A73 MP4, Mali-G72 MP3

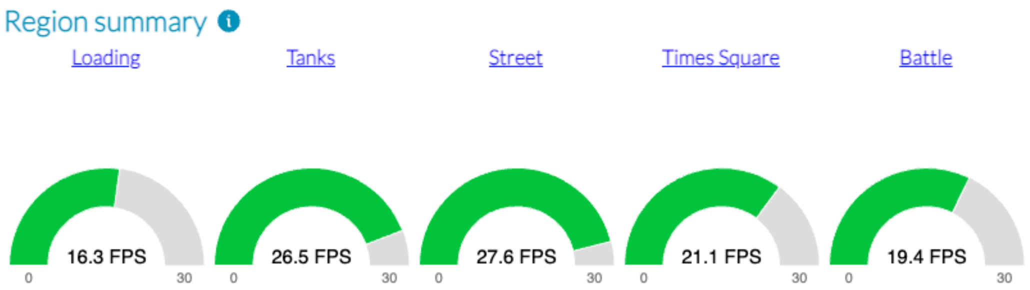
Capture summary

You are hitting your performance target for 13% of the time within your application. For the frames below target you are predominantly fragment bound. Read our optimization advice.

Where Performance Advisor has found a potential problem, you'll see advice above the chart, with a link to optimization advice on the Arm Developer website.

Region summary

If you have included region annotations, to provide context to different sections of your game, you will see the average frame rate achieved for each region.



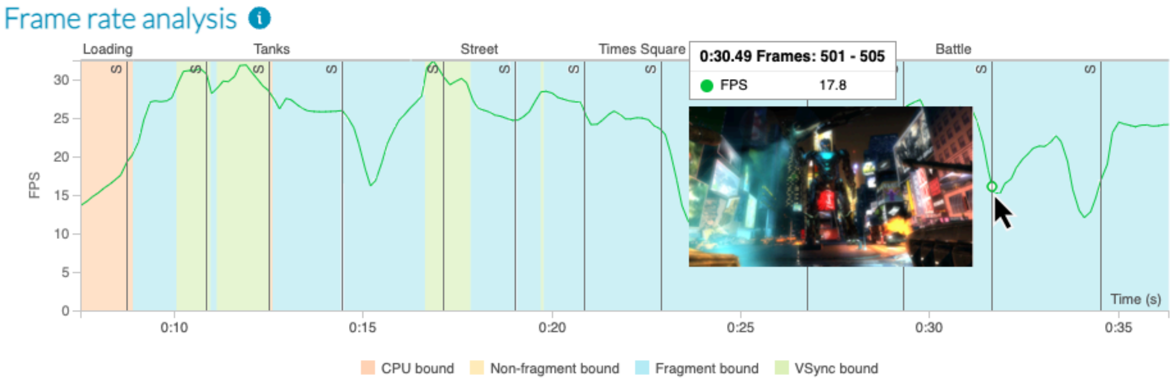
Region summary

Frame rate analysis

The frame rate analysis chart shows how the frame rate changes throughout the duration of your capture. Where the application is performing well, the background colour of the chart is green. In poorly performing sections, the background colour indicates which workload is causing the problem (CPU, fragment or non-fragment). Click and drag to zoom into a section of the chart for a closer look.

If you instructed Performance Advisor to collect slow frame screenshots, they are indicated with an S. Hover over the S markers to see the slow frame screenshots. To view the captured frame in high resolution, use the middle-click on your mouse, also known as scroll click.

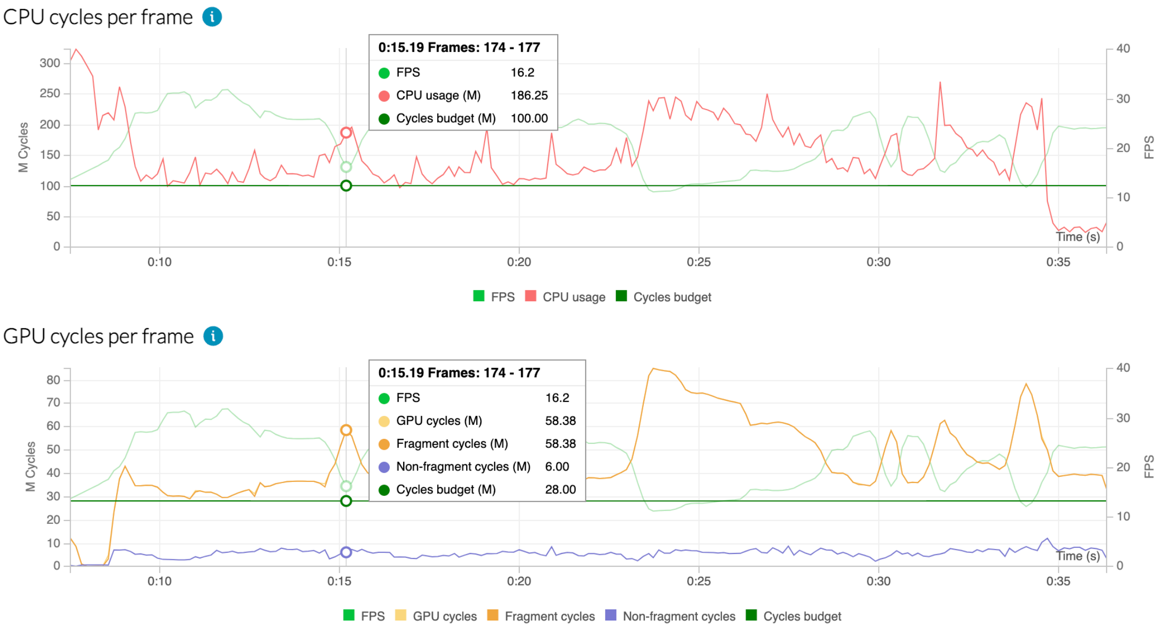
Figure 6-2: Frame rate analysis chart



CPU and GPU cycles per frame

Each chart also shows the FPS, so that you can look for areas of correlation that might indicate a problem. In this example, if we look at GPU cycles per frame, we can see that when the number of fragment cycles increases, FPS drops.

Figure 6-3: CPU and GPU cycles per frame charts



You can add your own expected performance budgets to each chart, to help you identify where the application has pushed the device too far. If you know the top frequency for the GPU in the device, and you have a target FPS, you can calculate the maximum allowable GPU cost per frame:

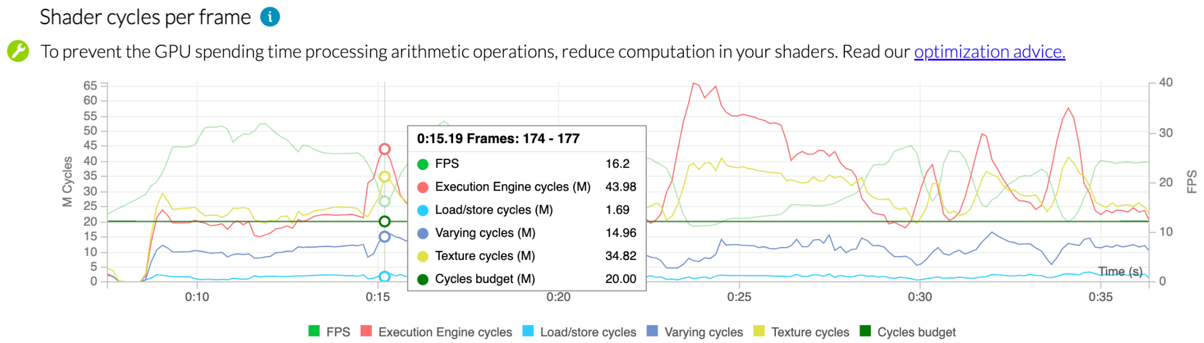
```
GPU maximum frequency / frame rate = maximum GPU cycles per frame
```

For details on how to add budgets to Performance Advisor report, refer to Setting performance budgets in the Performance Advisor user guide.

Shader cycles per frame

If we look at the number of shader cycles per frame, we can see correlation between high numbers of execution engine cycles, and drops in FPS.

Figure 6-4: Shader cycles per frame charts


Shader cycles per frame

To prevent the GPU spending time processing arithmetic operations, reduce computation in your shaders. Read our optimization advice.

Performance Advisor tells us that the GPU is busy with arithmetic operations, and that the shaders used could be too complex.

From here, we can click through to read optimization advice on the Arm Developer website, about how to reduce shader complexity to improve their performance.

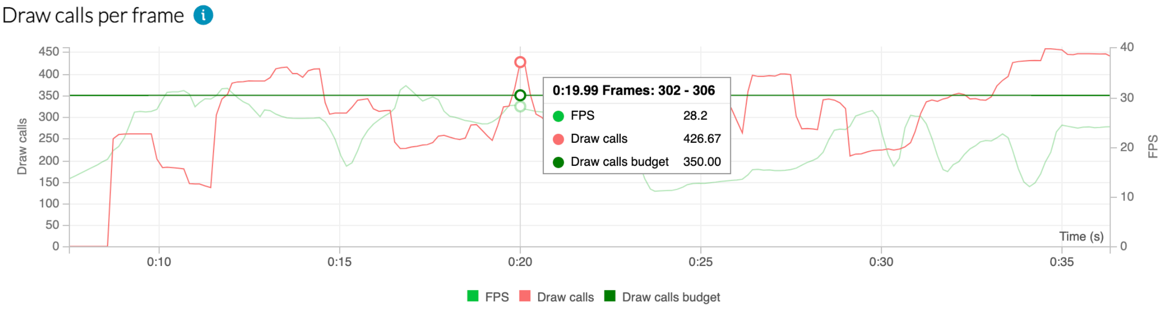
Tip

In many cases, problems like this can be solved by reducing shader precision to `mediump`, which can often have no visible impact to the scene.

Draw calls per frame

The draw calls per frame chart shows how many draw calls your application made over the duration of the capture. Draw calls are expensive for the CPU to process, so it is important to monitor the number of draw calls your application makes, and reduce them wherever possible.

Figure 6-5: Draw calls per frame charts



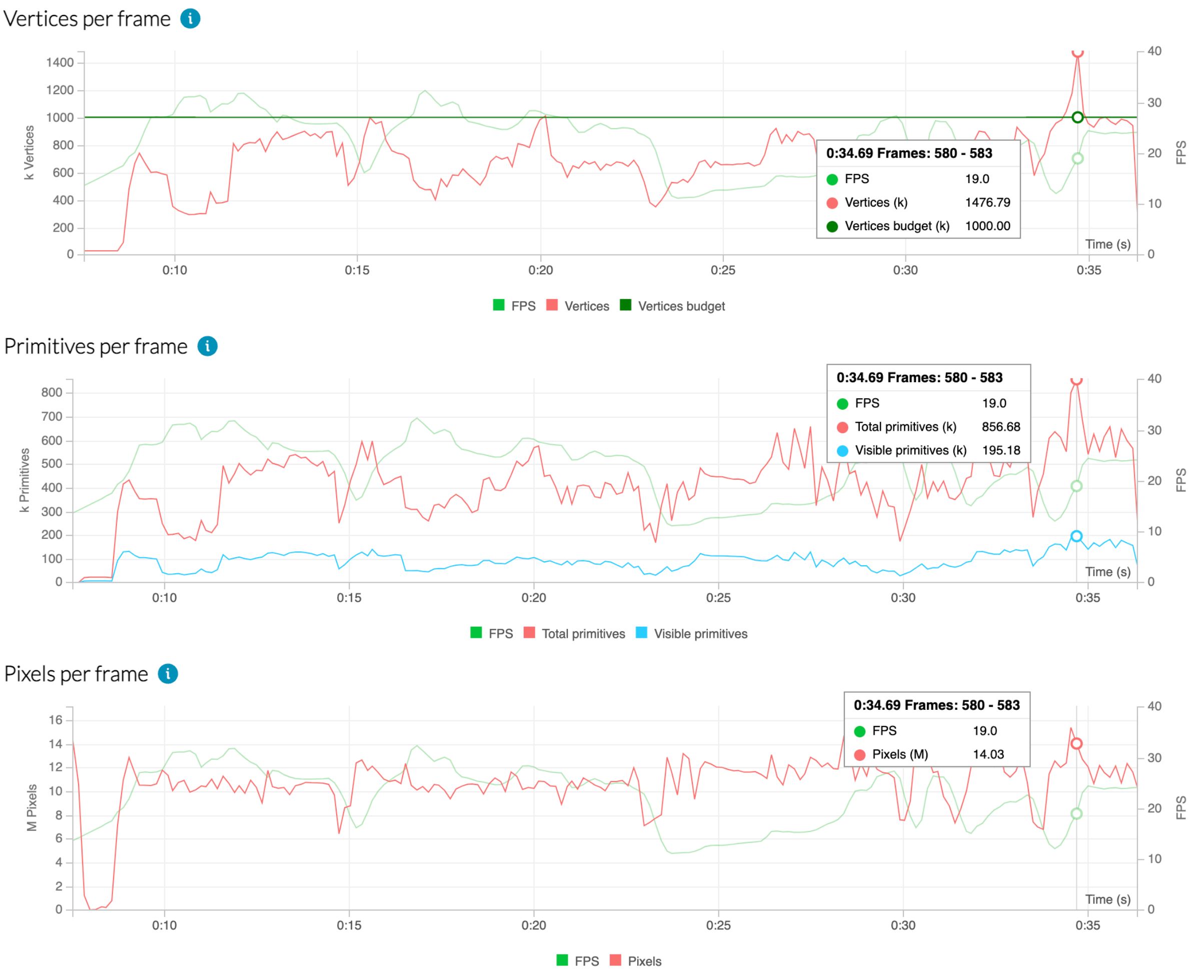
Tip

You can reduce the number of draw calls by batching objects into a single draw, and discarding draw calls that have no visible impact on the scene.

Vertices, primitives and pixels per frame

The vertices, primitives and pixels per frame charts help you to identify whether the game is requesting too much work from the GPU, or is requesting it inefficiently.

Figure 6-6: Vertices and primitives per frame charts



Vertices are expensive to process so aim to minimize them, and the stored data size and attribute precision of each vertex.

Ideally, the ratio of vertices to primitives should be close to 1:1, meaning that on average, each vertex is used by 3 primitives. If there are many more vertices than primitives, the object meshes used in the game could be improved, by identifying shared vertices and using an efficient indexing method to order them.

For primitives, we can see the total number processed, and also the number that were actually visible on-screen. We expect around half of the total number to be discarded, due to primitives that are in-frustum but back-facing. Compare these numbers, to check if you have a problem with objects being sent to the GPU but not contributing to the final scene.

Tip

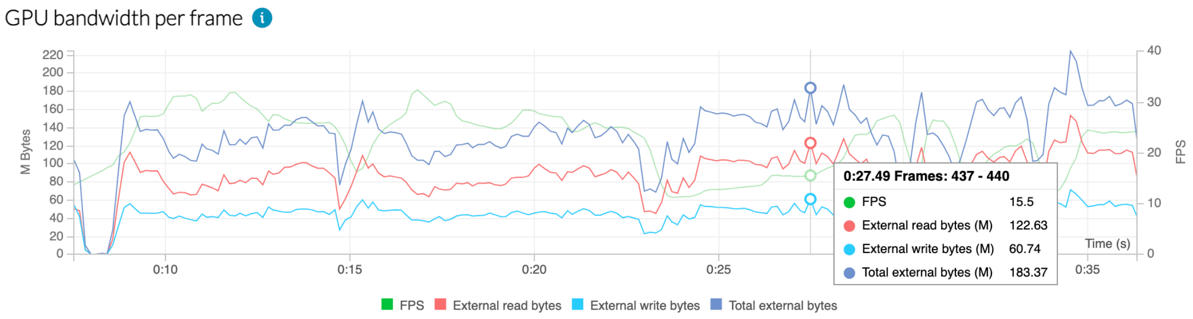
Because half of the primitives inside the frustum are back-face culled, for 3D content, approximately 50% of input primitives should be visible. If more than 50%

of primitives are culled, check the Mali geometry culling rate chart in Streamline to see why.

GPU bandwidth per frame

To stay within our power budget, we can monitor the distribution of GPU bandwidth, and the breakdown between external reads and writes. Even if the device can comfortably handle the memory demands of the game, we must try to minimize external memory accesses to reduce energy consumption. This frees up power budget for use elsewhere.

Figure 6-7: GPU bandwidth per frame charts



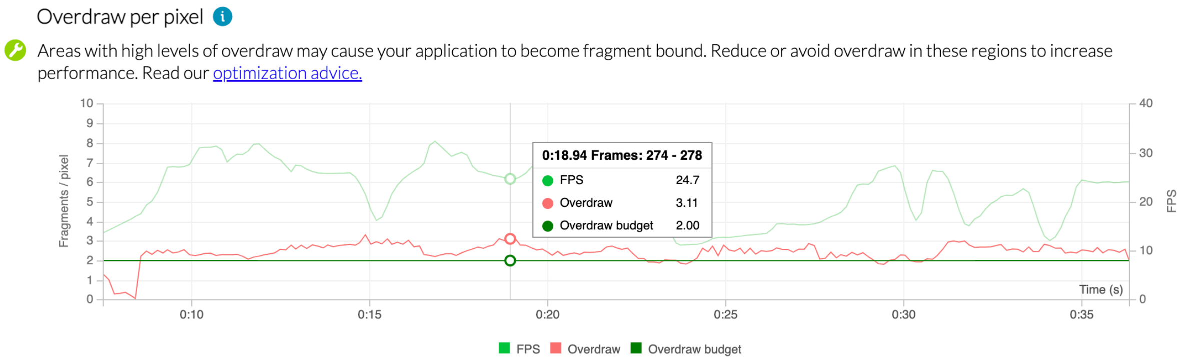
Tip

A good rule of thumb for memory cost, including memory bus, memory controller, and the RAM itself, is around 100 mW per GB/s of bandwidth used.

Overdraw per frame

Content that has a high degree of overdraw - multiple fragments shaded per output pixel - can suffer from poor performance because of the cumulative cost of shading all of the layers. Here you can monitor whether you have an acceptable level of overdraw.

Figure 6-8: Overdraw per frame chart



Areas with high levels of overdraw may cause your application to become fragment bound. Reduce or avoid overdraw in these regions to increase performance. Read our optimization advice.

Next Steps

Generating a performance report as a one-off can give you a snapshot of performance. But what if you could generate these reports automatically, every night, so that you team has timely performance data after every nightly build of your application?

Performance Advisor can be used as part of a continuous integration workflow, to generate both HTML and JSON reports automatically. Refer to Integrate Arm Mobile Studio into a CI workflow for details on how to do this.

7. Generate a JSON report

Performance Advisor can generate report data in JSON format, so that you can import it into any JSON-compatible database, and visualize the data in your own performance dashboards, to track performance over multiple runs.

Procedure

1. Run the streamline_me.py script and capture a Streamline profile, in the same way as is described for HTML reports.
2. Run the `Streamline-cli -pa` command on your Streamline capture file, with the `--type=json` option to generate a JSON report. In this example, the JSON report will be named `report.json`:

   ```
   Streamline-cli -pa --type=json:report.json my_capture.apc
   ```

   You can run one command and specify both HTML and JSON types, to generate a report in each format:

   ```
   Streamline-cli -pa --type=html:report.html,json:report.json my_capture.apc
   ```

   The JSON report is saved to the current location.

Tip

You can also generate a report that compares two JSON reports and returns the differences in performance metrics. Refer to the user guide for instructions on how to to this.

Next steps

For more information about how to use JSON files to monitor data over time, refer to Integrate Arm Mobile Studio into a CI workflow.

8. Command-line options

The connection script `streamline_me.py` and the `Streamline-cli -pa` command both accept a range of command-line options to control how they run. Refer to the following topics in the Performance Advisor user guide for full descriptions of these options:

- streamline_me.py command-line options
- Streamline-cli -pa command-line options

9. Troubleshooting Performance Advisor reports

In this section you will find answers to common problems that might occur when running the `streamline_me.py` script, capturing data from your device with Streamline, or generating a Performance Advisor report.

My computer can not access my device

When attempting to run the `streamline_me.py` script the following error is returned:

```
ERROR: Device must be connected; none available
```

Try the following to resolve this:

1. Check that the device is connected to your computer via USB.
2. Check that the device is is set to Developer mode.
3. Check that the device has USB debugging enabled in Settings > Developer options. When enabling USB debugging, your device may ask you to authorize connection to your computer. Confirm this.
4. Ensure you have installed Android Debug Bridge (ADB).
5. In a shell terminal, run the `adb devices` command. This command returns the ID of all connected devices. For example, with one device connected:

   ```
   adb devices
   List of devices attached
   RZ8MC03VVEW device
   ```

   If the device is listed as unauthorized, this means that your device has USB debugging enabled, but the computer it is connected to has not been given authority to access it.
6. Go to Settings > Developer Options and disable and re-enable USB debugging. You can also try revoking USB debugging authorizations here. When re-enabling USB debugging, your device should ask you to authorize access from your computer.

Can not run the Streamline-cli -pa command

Performance Advisor returns a ‘command not found’ error when trying to generate a report.

Check that either:

- You are currently in the Performance Advisor directory where the `Streamline-cli` executable is located:

  ```
  /<installation_directory>/streamline
  ```

- You have set your `PATH` environment variable to include the path to the Streamline directory, so that you can run the `streamline-cli -pa` command from any directory. Check the steps in the Setup tasks section for instructions on how to do this.

Tip

You can test whether the `streamline-cli -pa` command can be accessed, by typing `Streamline-cli -pa --help` to display the command documentation.

Performance Advisor can not find frames

When generating a report, Performance Advisor fails with the following error:

```
ERROR: Cannot find any frames. Please refer to the user guide for methods of
 providing frame data to Performance Advisor.
```

This can occur if the Streamline capture does not contain the frame data that Performance Advisor needs to generate a report.

You must use the `streamline_me.py` script to set up the device before capturing a profile with Streamline, as described in Run the streamline_me.py script.

From Arm Mobile Studio version 2023.2, the `--lwi-mode=counters` option enables frame boundary instrumentation. This option is required unless you have manually included frame boundary annotations in your application. Check that you have used this option when running the `streamline_me.py` script. For more information about using manual annotations, refer to the Performance Advisor user guide

Screenshots can not be found

After specifying to collect screenshots when running the `streamline_me.py` script, and generating a report with Performance Advisor, no screenshots are included in the report, and Performance Advisor reports the following warning message:

```
WARNING: No screenshots found in <your_directory>. If this is not expected, check
 the interceptor was used correctly.
```

Try the following to resolve this issue:

1. The directory where screenshots are saved is specified:
   - With the `--lwi-out-dir` option when running the `streamline_me.py` script.
   - With the `--frame-capture` option when generating the report with the `streamline-cli -pa` command.

   Check that you specified the correct directory when using these options.
2. Screenshots are only saved to the output directory you specified, when you stop the `streamline_me.py` script.

Go to the terminal window where you ran the `streamline_me.py` script and press Enter to stop it. If any screenshots were collected, they will be saved to the location you specified with the `--lwi-out-dir` option. You can now try running the `streamline-cli -pa` command again to generate the report.

Next steps

Refer to the Arm Mobile Studio FAQs for more troubleshooting topics.

10. Related information

Here are some resources related to material in this guide:

- More information about Performance Advisor on the Arm Developer website
- Mali GPU training video - Performance Advisor
- Setting performance budgets with Performance Advisor
- Optimization advice for mobile applications
- Integrate Arm Mobile Studio into a CI workflow
- Performance Advisor user guide