Performance Advisor
User Guide

Copyright © 2020 Arm Limited or its affiliates. All rights reserved.

Release Information

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100-00</td>
<td>28 February 2020</td>
<td>Non-Confidential</td>
<td>New document for v1.0</td>
</tr>
</tbody>
</table>

Non-Confidential

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, third party 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.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. 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 subsidiaries) 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 [http://www.arm.com/company/policies/trademarks](http://www.arm.com/company/policies/trademarks).

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.


110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349
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.

Web Address
www.arm.com
Contents
Performance Advisor User Guide

Preface
About this book .......................................................... ................................. 7

Chapter 1 Introduction to Performance Advisor
1.1 Overview of Performance Advisor ................................................................. 1-10
1.2 Performance Advisor workflows .................................................................... 1-12

Chapter 2 Before you begin
2.1 Download and install Arm® Mobile Studio ..................................................... 2-15
2.2 Integrate Performance Advisor with your application .................................... 2-16

Chapter 3 Quick start guide
3.1 Connect Streamline to your device ................................................................. 3-22
3.2 Choose a counter template ............................................................................. 3-23
3.3 Capture a Streamline profile .......................................................................... 3-25
3.4 Generate a performance report ....................................................................... 3-27

Chapter 4 Running Performance Advisor in continuous integration workflows
4.1 Generate a headless capture .......................................................................... 4-30
4.2 Export performance data as a JSON file ......................................................... 4-31
4.3 Generate multiple report types ...................................................................... 4-34

Chapter 5 Capturing a slow frame
5.1 Capturing slow frame rate images ................................................................. 5-36

Copyright © 2020 Arm Limited or its affiliates. All rights reserved.
Non-Confidential
5.2 Tagging slow frames ................................................................. 5-38

Chapter 6 Adding semantic input to the reports
6.1 Manually create annotations from your application ......................... 6-40
6.2 Specify a CSV file containing the regions ......................................... 6-41

Appendix A Analytics
A.1 Analytics .................................................................................. Appx-A-43

Appendix B Command line options
B.1 The pa command ........................................................................ Appx-B-45
B.2 The lwi_me.py command ............................................................ Appx-B-47
This preface introduces the *Performance Advisor User Guide*. It contains the following:

- *About this book on page 7.*
About this book

This book describes how to install and use Arm® Performance Advisor to generate reports from your Arm Streamline capture data.

Using this book

This book is organized into the following chapters:

**Chapter 1 Introduction to Performance Advisor**
This section introduces the Performance Advisor tool and the workflows that it is designed to handle.

**Chapter 2 Before you begin**
Follow the steps in this section to set up Arm Mobile Studio and integrate Performance Advisor with your application.

**Chapter 3 Quick start guide**
Performance Advisor runs on a capture file generated from Streamline. Follow the steps in this section when you are ready to perform an interactive capture.

**Chapter 4 Running Performance Advisor in continuous integration workflows**
Gain insights into how your application is performing, and learn which problem areas to focus on, by embedding performance analysis into your continuous integration workflow. Compare performance of your application with specific metrics using the Performance Advisor report then, if you require a deeper analysis, analyze further using Streamline.

**Chapter 5 Capturing a slow frame**
Use the lightweight interceptor (LWI) in different modes to identify slow frames. Before you can use the LWI, you must first integrate it with your application.

**Chapter 6 Adding semantic input to the reports**
Performance Advisor can use semantic information that the application provides as key input data when generating the analysis reports.

**Appendix A Analytics**

**Appendix B Command line options**

Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm® Glossary for more information.

Typographic conventions

*italic*
Introduces special terminology, denotes cross-references, and citations.

**bold**
Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`
Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace
Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

monospace italic
Denotes arguments to monospace text where the argument is to be replaced by a specific value.

monospace bold
Denotes language keywords when used outside example code.

<and>
Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>

SMALL CAPITALS
Used in body text for a few terms that have specific technical meanings, that are defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Feedback

Feedback on this product
If you have any comments or suggestions about this product, contact your supplier and give:
• The product name.
• The product revision or version.
• An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content
If you have comments on content then send an e-mail to errata@arm.com. Give:
• The title Performance Advisor User Guide.
• The number 102009_0100_00_en.
• If applicable, the page number(s) to which your comments refer.
• A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note
Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Other information
• Arm® Developer.
• Arm® Information Center.
• Arm® Technical Support Knowledge Articles.
• Technical Support.
• Arm® Glossary.
This section introduces the Performance Advisor tool and the workflows that it is designed to handle.

It contains the following sections:

- 1.1 Overview of Performance Advisor on page 1-10.
- 1.2 Performance Advisor workflows on page 1-12.
1.1 Overview of Performance Advisor

Performance Advisor analyzes performance data from your Arm Streamline capture, and generates a report that shows how your application is performing on your mobile device.

The summary at the top of the report shows whether your application is vertex or fragment bound. See how efficiently your CPU and GPU are running, your boundness split, and whether you are achieving your required frame rate.

To help you further understand how your application is performing over time, you can analyze key metrics shown on a series of charts:

**Overdraw per pixel**
Identify problems caused by transparency or rendering order, by monitoring the number of times pixels are shaded before they are displayed.

**Draw calls per frame**
To identify CPU workload inefficiencies, check the absolute number of draw calls per frame.

**Primitives per frame**
See how many input primitives are being processed per frame, and how many of them are visible in the scene.

**Pixels per frame**
See the total number of pixels being rendered per frame. This metric helps you to rule out problems caused by changes in the application render pass configuration, such as additional passes for new shadow casters or post-processing effects.

**Shader cycles per frame**
The total number of shader cycles per frame, broken down by pipeline, so that you can see which workloads are occupying the GPU.

**GPU cycles per frame**
See how the GPU is processing vertex and fragment workloads, and whether the shader core resources are balanced.

**GPU bandwidth per frame**
Monitor the distribution of GPU bandwidth, including the breakdown between reads and writes, so that you can minimize external memory accesses to save energy.
Running the Performance Advisor report regularly enables you to get performance feedback throughout the development cycle. You can also integrate Performance Advisor in your performance regression workflows, by generating machine-readable JSON reports that you can import into other tracking systems.

Related references
Chapter 2 Before you begin on page 2-14
Chapter 3 Quick start guide on page 3-21
1.2 Performance Advisor workflows

You can use Performance Advisor with Streamline in several different workflows, enabling you to solve multiple different types of problem.

Interactive capture with Performance Advisor report

You can use Performance Advisor to assist with a manual debug session. Manually connect to a target and capture data using Streamline. Use Performance Advisor to post-process the dataset to provide an initial quick analysis.

Automated capture with Performance Advisor report

Note

This feature is only available with an Arm Mobile Studio Professional license. For more information, email mobilestudio@arm.com.

You can use Streamline and Performance Advisor as part of a continuous integration (CI) workflow. To capture data from automated game tests, without using the Streamline GUI on the host, integrate the gator daemon from Streamline into a nightly test system. Use Performance Advisor to generate a report, which can be published automatically. This workflow enables a QA team to review the status each morning.
Automated capture with Performance Advisor data export

You can use Streamline and Performance Advisor to generate a machine-readable JSON report. You can import data from the JSON report into other QA test reporting systems, allowing automated regression tracking of in-depth workload metrics.

Using Streamline and Graphics Analyzer for further deep-dive analysis

The Performance Advisor report shows where your application is causing a problem. You can then use the other tools in Arm Mobile Studio suite to investigate any problems in more detail.

Streamline

Capture a profile of your application running on a mobile device and see where your system spends most of its time. Use interactive charts and comprehensive data visualizations to identify whether any performance bottlenecks are being caused by CPU processing or GPU rendering.

Graphics Analyzer

Helps OpenGL ES and Vulkan developers to get the best out of their applications, and find attempted misuse of the API. Analyze the effect of individual API calls by looking at API call arguments and return values in your target application.

The APC data file that the CI workflow creates is a full Streamline capture that you can import into the Streamline GUI. Arm recommends that you store the APC data file alongside other build artifacts. If Performance Advisor reports a problem, it is then immediately available for manual investigation in Streamline.

Note

For more information about using Streamline for profiling graphical applications running on Mali GPUs, see the Arm Community blog Accelerating Mali GPU Analysis using Arm Mobile Studio.
Chapter 2
Before you begin

Follow the steps in this section to set up Arm Mobile Studio and integrate Performance Advisor with your application.

It contains the following sections:
• 2.1 Download and install Arm® Mobile Studio on page 2-15.
• 2.2 Integrate Performance Advisor with your application on page 2-16.
2.1 Download and install Arm® Mobile Studio

Before you can use Performance Advisor, download and install the Arm Mobile Studio suite.

Note

A list of the recommended devices that support Arm Mobile Studio is available from https://developer.arm.com/tools-and-software/graphics-and-gaming/arm-mobile-studio/support/supported-devices.

Procedure


Next Steps

See 2.2 Integrate Performance Advisor with your application on page 2-16 for information about using the lightweight interceptor (LWI) and the Graphics Analyzer interceptor for capturing information from your test application.
2.2 Integrate Performance Advisor with your application

Performance Advisor collects performance data, such as frame rate and API call counts, from your application. To enable Performance Advisor to collect this data, you must package the lightweight interceptor library (LWI) with the application. This library is provided in your Arm Mobile Studio installation directory.

The LWI enables you to capture performance data automatically from your application, such as frame rate and frame captures. It is a lighter version of the Graphics Analyzer interceptor.

The LWI enables you to automatically capture data in the following situations:

• To automatically detect frame boundaries, or other API statistics, instead of manually embedding frame markers into the application.

• To identify slow parts of your application, you can capture a screenshot when your application goes below a threshold value that you configure.

Note

The Graphics Analyzer interceptor enables you to measure the frame rate of an application to determine its performance, and intercept graphics API calls.

Integrating the Graphics Analyzer interceptor causes higher target CPU overhead during profiling, and does not produce screenshots for the report. However, it enables you to use Graphics Analyzer with the same application build, without having to change the interceptor.

To integrate the Graphics Analyzer interceptor, follow the steps in the process most suitable for your application:

• For Unity applications, see Arm Guide for Unity Developers.

• For applications not using Unity, see Getting started with Graphics Analyzer.

To capture frames when the frame rate goes below a specified value, you must use the lightweight interceptor instead.

OpenGL ES

For OpenGL ES applications, package the required library file libMGD.so, which is provided in your Arm Mobile Studio package:

<install_directory>/performance_advisor/lwi/target/android/arm/unrooted/

Two versions of the library are provided:

• For 64-bit targets, use the library file located in the arm64-v8a directory.

• For 32-bit targets, use the library file located in the armeabi-v7a directory.

Note

You can package one or both interceptor libraries depending on the requirements of your application.

Vulkan

For Vulkan applications, package the required Vulkan layer file, which is provided in your Arm Mobile Studio package:

<install_directory>/performance_advisor/lwi/target/android/arm/rooted/

Note

If your target device is running Android 9 or above, you do not need to package the Vulkan layer with the application. Instead specify the path to the Vulkan layer when running the target connection script.
Two versions of the library are provided:
• For 64-bit targets, use the library file located in the arm64-v8a directory.
• For 32-bit targets, use the library file located in the armeabi-v7a directory.

**Prepare your Unity project**
1. Copy the required `libMGD.so` file or Vulkan layer into the `Assets/Plugins/Android/` directory in your Unity project. Create this directory if it does not exist.

If you are packaging both interceptor libraries:
• Create two directories in the `Assets/Plugins/` directory. For example, `armv7` and `armv8`.
• Create a directory called `Android` in each of these directories.
• Copy each `libMGD.so` file into the appropriate `Android` directory.
2. Select the library in Unity and set the following attributes in the Inspector:
   • Under **Select platforms for plugin**, select **Android**.
   • Under **Platform settings**, set the **CPU** architecture to **ARM64** for 64-bit applications, or **ARMv7** for 32-bit applications.
   • Click **Apply**.

3. Select **File > Build Settings**, then select **Player Settings**.
4. Under **Identification**, set **Target API Level** to the required Android version.

   __________ Note __________

   By default, **Target API Level** is set to the latest version of the Android SDK tools that you have installed. If you change to a lower API level, ensure that you have the SDK tools for that version installed. If you build for a higher API version later, change this setting accordingly.

5. Under **Configuration**, set the following options to build a 64-bit application:
   a. Set the scripting backend in Unity to work with 64-bit targets. Set **Scripting Backend** to **IL2CPP**. For more information about IL2CPP, refer to the Unity documentation.
   b. Under **Target Architectures**, select **ARM64**.

To build a 32-bit application:
- Leave the scripting backend at its default setting, **Mono**.
- Under **Target Architectures**, select **ARM7**.

6. Close the **Player Settings**. In the **Build Settings**, select the **Development Build** checkbox. This option ensures that your application is marked as debuggable in the Android application manifest.
7. To build your APK and install it on your device in one step, select **Build and Run**. Alternatively, select **Build** to build the APK and then install it on your device using Android Debug Bridge:

```
adb install -r YourApplication.apk
```

**Prepare your Android Studio project**

For OpenGL ES applications, supply the path to the LWI library in your applications gradle file:

```groovy
android {
    sourceSets {
        main {
            jniLibs.srcDirs += 'install_directory/performance_advisor/lwi/target/android/arm/unrooted/
        }
    }
}
```

For Vulkan applications, supply the path to the Vulkan layer files in your applications gradle file:

```groovy
android {
    sourceSets {
        main {
            jniLibs.srcDirs += '<install_directory>/performance_advisor/lwi/target/
android/arm/rooted/
        }
    }
}
```

Load the library in a static block in your code:

```java
static {
    try {
        System.loadLibrary("LWI");
    } catch (UnsatisfiedLinkError e) {
        ...}
}
```

Build your APK and install it on your device.
Connect your device

When you have packaged the required interceptor file into your application, use the lwi_me.py script to set up your target device and run your test case. The script is located in the Arm Mobile Studio installation directory:

<install_directory>/performance_advisor/lwi/helpers/lwi_me.py

Note

For more information about the lwi_me.py command, see B.2 The lwi_me.py command on page Appx-B-47.

See Chapter 5 Capturing a slow frame on page 5-35 for more information about monitoring the frame rate and triggering frame captures.
Chapter 3
Quick start guide

Performance Advisor runs on a capture file generated from Streamline. Follow the steps in this section when you are ready to perform an interactive capture.

--- Note ---
If you already have the capture files, you can go straight to Generate a performance report on page 3-27.

It contains the following sections:
• 3.1 Connect Streamline to your device on page 3-22.
• 3.2 Choose a counter template on page 3-23.
• 3.3 Capture a Streamline profile on page 3-25.
• 3.4 Generate a performance report on page 3-27.
3.1 Connect Streamline to your device

Arm provides a Python script, `gator_me.py` that installs a daemon, `gatord`, on your device. Run the script so that Streamline can connect to unrooted Android devices, and collect data.

**Procedure**

1. On your host machine, navigate to the Streamline installation directory, `<install_directory>/streamline/gator/`.

2. To supply the path for the `gatord` binary that will be installed on the device, run the `gator_me.py` Python script with the `--daemon` option.
   
   Your installation directory contains two versions of `gatord`, for 32-bit or 64-bit architectures:
   
   - `<install_directory>/streamline/bin/arm/` for 32-bit architectures.
   - `<install_directory>/streamline/bin/arm64/` for Armv8 64-bit architectures.
   
   For example:
   ```
   python3 gator_me.py --daemon <install_directory>/streamline/bin/arm64/gatord
   ```

3. The script returns a numbered list of the Android package names for the debuggable applications that are installed on your device. Enter the number of the package you want to profile.
   
   _____ Note _____
   
   Alternatively, if you know the Android package name of the application that you want to profile, you can specify it when running the script, using the `--package` option.
   ```
   python3 gator_me.py --package <app.package.name> --daemon <install_directory>/streamline/bin/arm64/gatord
   ```

4. Launch Streamline:
   
   - On Windows, from the Start menu, navigate to Arm MS 2020.0 and select Arm MS Streamline 2020.0.
   - On macOS, go to the `<install_directory>/streamline` folder, and double-click the Streamline.app file.
   - On Linux, go to the `<install_directory>/streamline` folder, and run the Streamline file:
   ```
   cd <install_directory>/streamline
   ./Streamline
   ```

5. In the **Target** view, click ![Target icon](image) and select your device.

**Next Steps**

Choose a counter template. For more information about how to find and select a counter template, see 3.2 Choose a counter template on page 3-23.
3.2 Choose a counter template

Counter templates are pre-defined sets of counters that enable you to review the performance of both CPU and GPU behavior. Choose the most appropriate template for the GPU in your target device.

Procedure

1. To choose a counter template, in Target view, click Counter configuration .
2. Click Add counters from a template to see a list of available templates.

3. Select a counter template appropriate for the GPU in your target device, then Save your changes.
   - The number of counters in the template that your target device supports is shown next to each template. Choose the template with the highest number of supported counters. For example, here, 34 of the 38 available counters in the Mali Midgard template are supported in the connected device.

   ![Counter Configuration Screen](image)

   • After the debuggable package has been selected, the python scripts gator_me or lwi_me identifies the appropriate GPU family in the console window. For example:

     ```
     Searching for a Mali GPU:
     Mali-G72 GPU found
     ```

4. Optionally, in Target view, click Capture & analysis options to set more capture options, including the sample rate and the capture duration (by default unlimited). Refer to Capture options in the Arm Streamline User Guide.
Next Steps

Capture a profile using Arm Streamline. For more information about how to capture the behavior of your CPU and GPU performance using Arm Streamline, see 3.3 Capture a Streamline profile on page 3-25.
3.3 Capture a Streamline profile

Start a capture session to profile data from your application in real time. When the capture session ends, Streamline automatically opens a report for you to analyze later.

Prerequisites

- Install Python 3.6 (or higher). Arm Mobile Studio uses Python to run the provided gator_me.py script, which uses the gatord agent to connect Streamline to your Android target.
- Install Android Debug Bridge (adb). Arm Mobile Studio uses the adb utility to connect to the target device. Download the latest version of adb from the Android SDK platform tools (https://developer.android.com/studio/releases/platform-tools.html).
- If you are using Linux, edit your PATH environment variable to add the paths to the Performance Advisor, Python3, and Android SDK platform tools directories.
- Set your device to Developer Mode.
- Enable USB Debugging by selecting Settings > Developer options.
- Connect your device to the host machine through USB.
- Run the adb devices command on the host to check that connection is successful. If successful, this command returns the ID of your target.

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

- To capture performance data, configure the test application as a debuggable application:
  - If you are not using Unity, enable the android:debuggable setting in the application manifest file, as described in https://developer.android.com/guide/topics/manifest/application-element
  - In Unity, when building your application, select the Development Build option in Build Settings.

Procedure

1. Click Start Capture to start capturing data from the target device.
   Specify the name and location on the host of the capture file that Streamline will create when the capture is complete. Streamline then switches to Live view and waits for you to start the application on the device.
2. Start the application that you want to profile.
   Live view shows charts for each counter that you selected. Below the charts is a list of running processes in your application with their CPU usage. The charts now start updating in real time to show the data that gatord captures from your running application.
3. Unless you specified a capture duration, click Stop capture to end the capture.
   Streamline stores the capture file in the location that you specified previously, and then prepares the capture for analysis. When complete, the capture appears in the Timeline view.
4. IMPORTANT: Switch back to the terminal running the gator_me.py script and press any key to terminate it. The script kills all processes that it started and removes gatord from the target.

5. Click **Switch and manage templates** and select the same counter configuration template that you chose to create the capture.

![Counter Configuration Template](image)

**Next Steps**

- *Generate a performance report on page 3-27*
- To analyze performance with Streamline, see *Analyze your capture* in the *Arm Streamline User Guide*. 
3.4 Generate a performance report

Generate a HTML performance report from an existing Streamline capture.

Prerequisites
A Streamline capture file.

Procedure
1. Open a terminal in the directory containing your APC file.

Note
The APC file can be a zip file or an uncompressed .apc directory.

2. Run Performance Advisor using the following command:

```
pa <capture.apc.zip> -p <app.package.name> -d <optional output dir>
```

Note
- For more information about the `pa` command, see The `pa` command on page Appx-B-45
- To include build and device information in the report, include the `--build-name`, `--build-timestamp`, and `--device-name` command-line options.

Performance Advisor saves a HTML file in the directory that you specified. The file contains the results of the performance analysis, and links to advice on how to improve the performance. As an example, here is part of a report:

```
Performance Advisor Report
```

(The report contains various performance metrics and recommendations.)
The summary section shown at the top of the report is based on the duration of your capture. To take a closer look at a specific area of interest, click and drag the cursor over the region to select it.

New charts now enable you to analyze the data over the region you selected. Click anywhere on the chart when you are ready to go back to the original capture duration.

To get help on overcoming graphics problems and optimizing your application, click the advice links on the report.

**Related tasks**
4.2 Export performance data as a JSON file on page 4-31
4.3 Generate multiple report types on page 4-34

**Related references**
B.1 The pa command on page Appx-B-45

**Related information**
Optimization advice for mobile applications
Chapter 4
Running Performance Advisor in continuous integration workflows

Gain insights into how your application is performing, and learn which problem areas to focus on, by embedding performance analysis into your continuous integration workflow. Compare performance of your application with specific metrics using the Performance Advisor report then, if you require a deeper analysis, analyze further using Streamline.

It contains the following sections:
- 4.1 Generate a headless capture on page 4-30.
- 4.2 Export performance data as a JSON file on page 4-31.
- 4.3 Generate multiple report types on page 4-34.
4.1 Generate a headless capture

Generate a headless capture from the command line.

**Prerequisites**

Generate a configuration.xml file by performing one manual capture on the target device, as described in *Chapter 3 Quick start guide on page 3-21.*

**Procedure**

1. Open a terminal and change to the directory that you created during the manual capture session.
2. Run the following command:

   ```
   python3 gator_me.py --package <app.package.name> --headless myCapture.apc.zip
   ```

   This command pushes gatord onto the target, ready to profile the target application.
3. When the script prints a prompt to do so, start the application on the target device.
4. To stop profiling, exit the application in one of the following ways:
   - Exit the application at the end of the test scenario.
   - Manually exit the application using the Android user interface.
   - Forcefully kill the application using:

   ```
   adb shell am force-stop <app.package.name>
   ```

   The script downloads the capture data to the host machine path specified by the --headless command-line option.

   **Note**

   Instead of exiting the application, you can specify a --headless-timeout <seconds> value. This method is not ideal for test scenarios with variable performance.
4.2 Export performance data as a JSON file

Generate a JSON report that you can import into other tools. Use reports from multiple test runs to track performance over time.

Note

JSON reports are only available with an Arm Mobile Studio Professional license.

JSON reports provide a raw data export that you can import into other tools, such as a NoSQL database, to compare different test runs. For example, you can track the average number of visible primitives per frame between builds.

Procedure

1. Open a terminal in the directory containing your APC file.

Note

The APC file can be a zip file or an uncompressed .apc directory.

2. Run Performance Advisor using the following command:

   \[ \text{pa } <\text{capture.apc.zip}> -p <\text{app.package.name}> -d <\text{optional output dir}> -t \text{json} \]

   To change the output file name, append it to the -t argument using a colon:

   \[ -t \text{json:your_file_name.json} \]

   The following is an example of part of a JSON report:

   ```json
   {
   "targetInfo": {
   "build": null,
   "device": "Example board",
   "processors": "Cortex-A55 MP4, Mali-G72"
   },
   "allCapture": {
   "averageFrameRateFps": 19.4,
   "boundnessSplitPercentage": {
   "fragment": 0.0,
   "vertex": 0.0,
   "vsync": 0.0,
   "cpu": 98.5,
   "unknown": 1.5
   },
   "averageUtilizationPercentage": {
   "averageGpuUtilization": 19.0,
   "averageCpuUtilization": 62.7
   }
   },
   "fpsBoundness": {
   "frameRate": {
   "average": 19.4,
   "max": 21.1,
   "min": 17.9,
   "centiles": {
   "80": 20.0,
   "98": 21.1,
   "95": 20.7
   }
   },
   "vsync": {
   "target": 60,
   "percentageTimeUnderTarget": 100
   }
   },
   "overdrawPerPixel": {
   "overdraw": {
   "average": 0.3,
   "max": 0.4,
   "min": 0.1,
   "centiles": {
   ```
"80": 0.4,
"98": 0.4,
"95": 0.4
}
},
"gpuUsagePerFrame": {
  "vertexCycles": {
    "average": 1707767.6,
    "max": 2039630.8,
    "min": 776117.5,
    "centiles": {
      "80": 1917112.6,
      "98": 2039630.8,
      "95": 2039630.8
    }
  },
  "gpuCycles": {
    "average": 4157114.0,
    "max": 4897026.6,
    "min": 1587167.6,
    "centiles": {
      "80": 4649032.8,
      "98": 4897026.6,
      "95": 4897026.6
    }
  },
  "fragmentCycles": {
    "average": 2449346.8,
    "max": 2911080.0,
    "min": 688366.8,
    "centiles": {
      "80": 2857394.4,
      "98": 2911080.0,
      "95": 2911080.0
    }
  }
},
"drawCallsPerFrame": {
  "drawCalls": {
    "average": 456.0,
    "max": 456.0,
    "min": 456.0,
    "centiles": {
      "80": 456.0,
      "98": 456.0,
      "95": 456.0
    }
  }
},
"primitivesPerFrame": {
  "totalPrimitives": {
    "average": 290318.2,
    "max": 331233.8,
    "min": 114309.3,
    "centiles": {
      "80": 325304.5,
      "98": 331233.8,
      "95": 331233.8
    }
  },
  "visiblePrimitives": {
    "average": 89856.7,
    "max": 102210.2,
    "min": 34685.2,
    "centiles": {
      "80": 100151.9,
      "98": 102210.2,
      "95": 102210.2
    }
  }
},
"pixelsPerFrame": {
  "pixels": {
    "average": 4669783.4,
    "max": 5315129.7,
    "min": 3197000.8,
    "centiles": {
      "80": 5165539.5,
      "98": 5315129.7,
      "95": 5315129.7
    }
  }
}
Related tasks
3.4 Generate a performance report on page 3-27
4.3 Generate multiple report types on page 4-34

Related references
B.1 The pa command on page Appx-B-45
4.3 Generate multiple report types

Generate a HTML performance report and a JSON performance report from an existing Streamline capture.

Prerequisites
A Streamline capture.

Procedure
1. Open a terminal in the directory containing your APC file.

   Note
   The APC file can be a zip file or an uncompressed .apc directory.

2. Run Performance Advisor using the following command:

   pa <capture.apc.zip> -p <app.package.name> -d <optional output dir> -t html,json

   To change the output file names, append each file name to the corresponding type argument using a colon:

   -t html:your_file_name.html,json:your_file_name.json

Related tasks
3.4 Generate a performance report on page 3-27
4.2 Export performance data as a JSON file on page 4-31

Related references
B.1 The pa command on page Appx-B-45
Chapter 5
Capturing a slow frame

Use the lightweight interceptor (LWI) in different modes to identify slow frames. Before you can use the LWI, you must first integrate it with your application.

It contains the following sections:
• 5.1 Capturing slow frame rate images on page 5-36.
• 5.2 Tagging slow frames on page 5-38.
5.1 Capturing slow frame rate images

Use Performance Advisor to continuously monitor frame rate and trigger frame captures when a slow part is detected.

Frame capture is not yet supported in Vulkan applications.

Procedure

1. Trace your application and output the capture to a specified folder.
   - For example, to trace an OpenGL ES application with default LWI parameters and place the LWI capture in a folder called /some/folder:
     ```
     python3 lwi_me.py -D <path to gatord> -P <app.package.name> --lwi on -o /some/folder
     ```
   - To specify a threshold value and capture a frame when the frame rate goes below 50fps:
     ```
     python3 lwi_me.py -D <path to gatord> -P <app.package.name> --lwi on --thr 50 -o /some/folder
     ```
   - To capture a frame when the frame rate goes below 50fps, and allow at least 100 frames between captures:
     ```
     python3 lwi_me.py -D <path to gatord> -P <app.package.name> --lwi on --thr 50 --gap 100 -o /some/folder
     ```

   -------- Note --------
   Capturing frames can affect performance. If you notice the a decrease in performance when capturing images, tag the slow frames instead. See 5.2 Tagging slow frames on page 5-38 for more information.
   --------

2. Manually capture a Streamline profile, as described in 3.3 Capture a Streamline profile on page 3-25.
   You do not need to run gator_me as it is called by the lwi_me script.

   -------- Note --------
   During the Streamline capture, the captured resources are written in the target when the trace reaches the end frame. The default is to end the capture at frame 500. You can adjust the end frame by specifying an alternative value for the FRAMEEND parameter of the lwi_me.py script.
   --------

3. To export the capture to the HTML report, send the frame capture path to the output directory:
   ```
   pa -c [capture.apc] -p <app.package.name> -f [frame_capture_folder]
   ```
   For more information about generating a HTML report, see Generate multiple report types on page 4-34.
   To see the captured frame, hover the cursor over the screen capture icon 5.
5 Capturing a slow frame

5.1 Capturing slow frame rate images
5.2 Tagging slow frames

If capturing frames directly impacts the performance of your application by reducing the frame rate, run the lwi_me.py command to capture the frame numbers in tag mode. Then run the lwi_me.py command to capture the frames in replay mode.

**Procedure**

1. Trace your application and output the capture to a specified folder.
   
   For example, use the following command to trace an OpenGL ES application, tagging a frame when the frame rate goes below 50fps:
   
   ```bash
   python3 lwi_me.py -D <path to gator> -P <app.package.name> --lwi on -Th 50 -M tag --lwi-mode tag -o /some/folder
   ```
   
   Run the file with tagged frame numbers using `--lwi-mode replay` to capture the tagged frames.
   
   ```bash
   python3 lwi_me.py -D <path to gator> -P <app.package.name> --lwi on -Th 50 -M tag --lwi-mode replay --lwi-slow-frames /some/folder/slow-frames -o /some/folder
   ```
   
2. Manually capture a Streamline profile, as described in 3.3 Capture a Streamline profile on page 3-25.

   **Note**
   
   During the Streamline capture, the captured resources are written in the target when the trace reaches the end frame. The default is to end the capture at frame 500. You can adjust the end frame by specifying an alternative value for the `FRAMEEND` parameter of the lwi_me.py script.

3. To export the capture to the HTML report, send the frame capture path to the output directory:
   
   ```bash
   pa -c [capture.apc] -p <app.package.name> -f [frame_capture_folder]
   ```
   
   For more information about generating a HTML report, see 4.3 Generate multiple report types on page 4-34.

   To see the captured frame, hover the cursor over the screen capture icon.
Performance Advisor can use semantic information that the application provides as key input data when generating the analysis reports.

The analysis reports support the use of region annotations to give context to the different frame ranges in a test scenario. Manually add these annotations into the application code. Alternatively, if manually adding annotations is not possible, or for quick debugging and extra analysis, specify a CSV file containing the regions. Give Performance Advisor the path to the CSV file using the --regions argument.

It contains the following sections:

- 6.1 Manually create annotations from your application on page 6-40.
- 6.2 Specify a CSV file containing the regions on page 6-41.
6.1 Manually create annotations from your application

Streamline allows an application to manually create annotations, which can be used as an alternative source of frame boundary annotations.

Native code

The native C code to include for generating annotations is located in the `<ms_install>/streamline/gator/annotate/` directory.

Unity plug-in code

Source code for a proof-of-concept plug-in for Unity 2018.2, which provides C# bindings for the Streamline annotation functions, is located here:

[GitHub] ARM-software Mobile Studio Unity Plugin

6.1.1 Generating frame boundary annotations

The Graphics Analyzer interceptor automatically adds frame boundary annotations. However, for more flexibility over how the frames are defined, you can manually specify the annotations. For example, if you only want to track certain frames or if you have multiple contexts.

To implement frame boundary annotations manually, generate Streamline marker annotations matching the following regular expression format, where the number is a monotonically incrementing frame number:

\[ F(/d+) \]

For example:

F10
F11
F12

6.1.2 Generating region annotations

Region annotations group consecutive frames into a named group, allowing you to indicate different parts of your application. Performance Advisor performs extra analysis on each of these regions and produces a separate section for each region in the report, with specific advice.

To implement frame range region annotations, manually generate Streamline Marker annotations matching the following regular expression format, where the label is a unique name in the test scenario:

\[ Region \text{ Start} (.*) \]  
\[ Region \text{ End} (.*) \]

To ensure that conclusions are statistically significant, Arm recommends keeping regions and subregions relatively large. For example, at least two seconds in length.
6.2 Specify a CSV file containing the regions

If manually adding annotations is not possible, or for quick debugging and extra analysis, specify a CSV file containing the regions and use the \texttt{--regions} argument.

Create a CSV file using the following format, where each region is on a new line:

\begin{tabular}{|c|c|c|}
\hline
Region Name, & Start Time (ms), & End Time (ms) \\
\hline
\end{tabular}

For example:

\begin{tabular}{|c|}
\hline
Test Region,0,1000 \\
\hline
\end{tabular}

\begin{tabular}{|c|}
\hline
\end{tabular}

--- Note ---

Do not put spaces next to the commas.

---

Give Performance Advisor the path to the CSV file using the \texttt{--regions} argument.
Appendix A
Analytics

It contains the following section:
• A.1 Analytics on page Appx-A-43.
A.1 Analytics

Arm periodically collects anonymous information about the usage of our products to understand, and analyze, what components or features you are using. We use this information to improve our products and your experience with them.

Product usage analytics contain information such as system information, settings, and usage of specific features of the product.

The data that we collect through Performance Advisor is anonymous and does not include any personal information:

- Version and build number of Performance Advisor
- Session time using Performance Advisor
- Edition of license that you are using (Starter, Evaluation, or Professional)
- Operating system details, such as version number, platform, language, and architecture (for example 64-bit)
- Number of monitors
- Screen pixels per inch (PPI) and monitor resolutions
- CPU and GPU information
- Java version and memory
- Number, and type (HTML or JSON), of reports generated
- Number of headless captures used
- Number of captures containing user regions
- Number of captures containing overdraw, or draw call information
- Number of times screenshots were supplied
- Total number of errors that you encountered, reported by type of error.
- Total number of licensing errors (unsupported/unknown license)

Note

Analytics collection is enabled by default.

- Set the command-line argument --disable-analytics when running Performance Advisor to disable it for the current invocation.
- Alternatively, set the ARM_DISABLE_ANALYTICS environment variable to any nonzero value before running Performance Advisor to disable analytics collection for all invocations.
Appendix B
Command line options

It contains the following sections:

- B.1 The pa command on page Appx-B-45.
- B.2 The lwi_me.py command on page Appx-B-47.
B.1 The pa command

The pa command runs Performance Advisor on a capture.

Syntax

```
pa [OPTIONS] <capture.apc>
```

Options

- `<capture.apc>`
  Path to the capture APC directory.
- `--centiles=int[,int...]`
  Comma-separated integer values specifying the percentiles to calculate for each data series. Default = 80,90,95.
- `-d, --directory=path`
  Output directory path for the reports.
- `--disable-analytics`
  Disables sending any analytics data to Arm.
- `-f, --frame-capture=path`
  Path to the frame captures directory.
- `-h, --help`
  Shows command-line arguments and descriptions and exits.
- `-m, --main-thread=string`
  The name of the main render thread to analyze.
- `-p, --process=string`
  Name of the process to inspect.
- `--[no-]progress`
  Whether to display progress bars or not.
- `-r, --regions=file`
  Takes a CSV file containing custom regions to add to the report. Expects each line of the CSV file to be of the format `regionName,startMs,endMs`. See 6.2 Specify a CSV file containing the regions on page 6-41.
- `-t, --type=type[:file][,type[:file]...]`
  Comma-separated list of report types, where the type is one of:
  ```
  json
  JSON CI report
  html
  Interactive html report
  ```
  You can specify an output filename for each report.
- `-v, --vsync=int`
  The vsync or target framerate. Default = 60.
- `-V, --version`
  Prints version information and exits.

Options for report metadata:
--application-name=string
  The human readable name of the application being analyzed. For example, "Awesome Game". If
  the name contains whitespace, use quotes. This name becomes the report title. Default =
  "Performance Advisor Report".

--build-name=string
  The build name of your application. For example, nightly. fa34c92.

--build-timestamp=string
  The timestamp of your application build. For example, Thu, 22 Aug 2019 12:47:30.

--device-name=string
  The name of the device that is used to obtain the capture.
B.2 The lwi_me.py command

Run `python3 lwi_me.py -h` to see the possible options and their default values for the `lwi_me.py` command.

Syntax

```
python3 lwi_me.py [OPTIONS]
```

Options

```
--device DEVICE, -E DEVICE
    The target device name (default=auto-detected).

--package PACKAGE, -P PACKAGE
    The application package name (default=auto-detected).

--headless CAPTURE_PATH, -H CAPTURE_PATH
    Perform a headless capture, writing the result to the path CAPTURE_PATH (default=perform interactive capture).

--headless-timeout TIMEOUT, -T TIMEOUT
    Exit the headless timeout after the specified number of seconds (default=wait for process exit).

--config CONFIG, -C CONFIG
    The capture counter config XML file you want to use (default=None for interactive, configuration.xml for headless)

--daemon DAEMON, -D DAEMON
    The path to the gatord binary you want to use (default=gatord).

--no-clean-start
    Disable pre-run device cleanup (default=enabled).

--no-clean-end
    Disable post-run device cleanup (default=enabled).

--overwrite
    Overwrite an earlier headless output (default=disabled).

--verbose, -v
    Enable verbose logging (default=disabled).

--lwi LWI
    Enable or disable the LWI. Possible values are 'on', 'off' and 'alone'. The 'alone' mode bypasses gator. Default is 'off'.

--lwi-api LWIAPI
    The API to listen to. Possible values are 'gles' or 'vulkan'.

--lwi-vk-layer-name VKLAYERNAME
    The vulkan layer name.

--lwi-vk-layer-lib-path VKLAYERLIBPATH
    The vulkan layer library path.

--lwi-fps-window FPSWINDOW, -W FPSWINDOW
    Size (in frames) of the sliding window used for FPS calculation.

--lwi-fps-threshold FPSTHRESHOLD, -Th FPSTHRESHOLD
    Perform capture if FPS goes under this threshold.

--lwi-frame-start FRAMESTART, -S FRAMESTART
    Start tracking from frame number.
--lwi-frame-end FRAMEEND, -N FRAMEEND
End tracking at frame number.

--lwi-frame-gap FRAMEGAP, -G FRAMEGAP
Minimum number of frames between two captures.

--lwi-mode LWIMODE, -M LWIMODE
Can be 'capture', 'tag' or 'replay' ('c', 't' or 'r') indicating if we allow capturing, only tagging slow frames or replaying.

--lwi-slow-frames LWISLOWFRAMES
Path to a file containing the indices of slow frames (required in 'replay' mode). This file can be generated with 'tag' mode.

--lwi-out-dir OUTDIR, -o OUTDIR
Folder where the LWI capture is output.