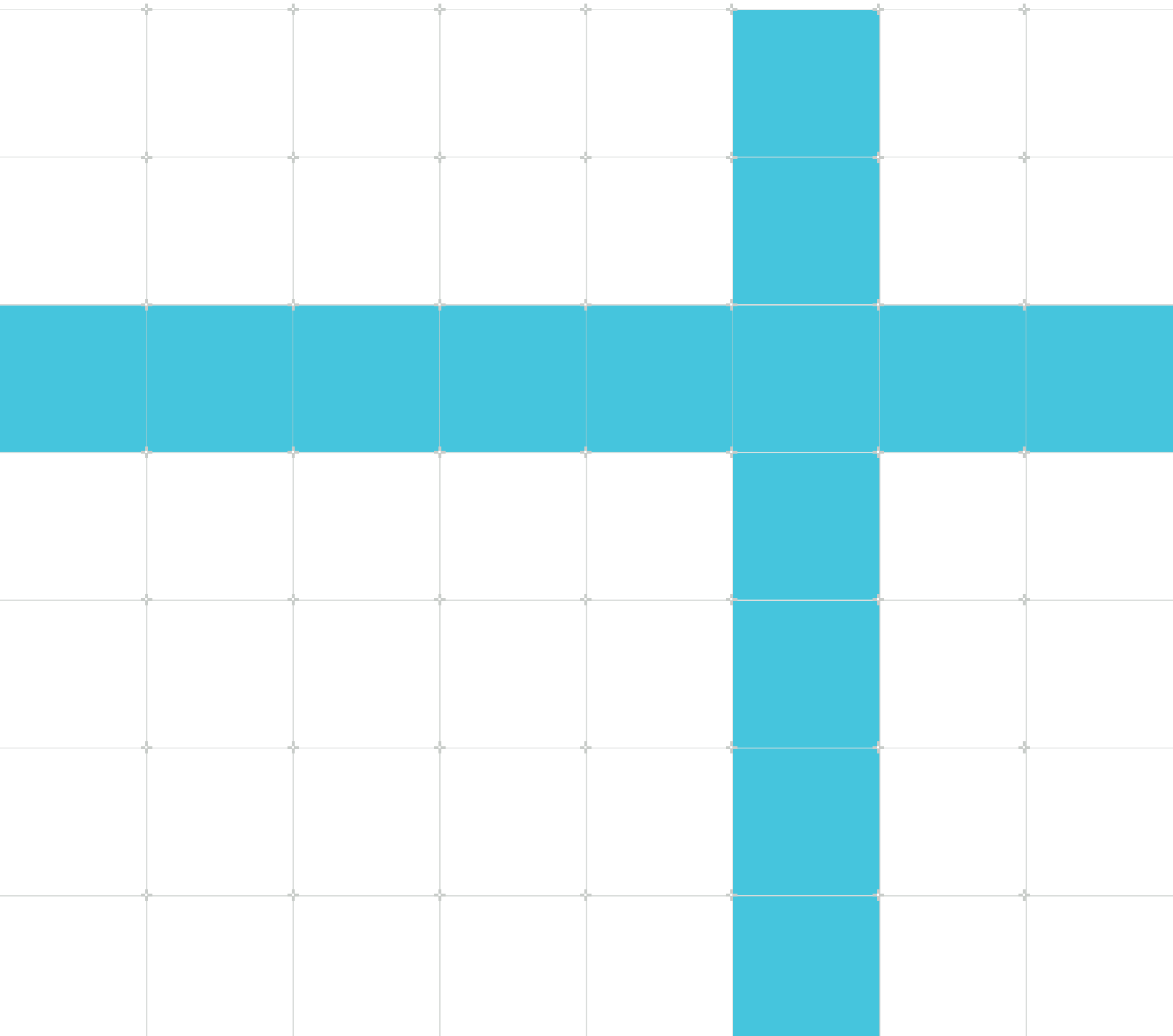
Get started with Mali Offline Compiler

Version 1.0

Non-Confidential

Issue 02
102468_0100_02_en



Get started with Mali Offline Compiler

Release information

Document history

Issue	Date	Confidentiality	Change
0100-00	12 March 2021	Non-Confidential	First release
0100-02	31 May 2022	Non-Confidential	Update to metadata

Proprietary Notice

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

This guide introduces how to use Mali Offline Compiler to analyze the performance of shader programs. This example demonstrates how to visualize performance bottlenecks on a Mali GPU target:

Video of whats new in Mali Offline Compiler 7.2

2. Before you begin

Before you learn how to use Mali Offline Compiler, you must install the Arm mobile studio and test the Mali Offline Compiler:

1. Download and install Arm Mobile Studio.
2. On Linux or macOS, add the install location of Mali Offline Compiler to your PATH environment variable so that you can compile from any directory. If you installed on Windows using the installer, this step is done automatically.

   If you omit this step, you must navigate to the `<install_location>/mali_offline_compiler` directory each time you want to run Mali Offline Compiler.
3. In a terminal, test that Mali Offline Compiler is installed correctly, by typing:

   ```
   malioc --help
   ```

   The `--help` option returns usage instructions and the full list of available options for the `malioc` command.

On macOS, Mali Offline Compiler might not be recognised as an application from an identified developer. To enable Mali Offline Compiler, open System Preferences > Security & Privacy, and select Allow Anyway for the malioc item.

3. Compile your shader

The following Open GL ES fragment shader implements the horizontal pass of a 5-tap separable Gaussian blur, with an optional tone mapping stage implemented using a matrix multiply:

```
#version 310 es
  #define WINDOW_SIZE 5

  precision highp float;
  precision highp sampler2D;

  uniform bool toneMap;
  uniform sampler2D texUnit;
  uniform mat4 colorModulation;
  uniform float gaussOffsets[WINDOW_SIZE];
  uniform float gaussWeights[WINDOW_SIZE];

  in vec2 texCoord;
  out vec4 fragColor;

  void main() {
     fragColor = vec4(0.0);

     // For each gaussian sample
     for (int i = 0; i < WINDOW_SIZE; i++) {
         // Create sample texture coord
         vec2 offsetTexCoord = texCoord + vec2(gaussOffsets[i], 0.0);

         // Load data and perform tone mapping
         vec4 data = texture(texUnit, offsetTexCoord);
         if (toneMap) {
             data *= colorModulation;
         }

         // Accumulate result
         fragColor += data * gaussWeights[i];
      }
  }
```

1. In a terminal window, enter the following command to instruct Mali Offline Compiler to compile the shader for a device with a Mali-G76 GPU:

   ```
   malioc -c Mali-G76 gauss_blur.frag
   ```

   This returns the following performance report:

   ```
   Mali Offline Compiler v7.0.0 (Build bc7a3e)
     Copyright 2007-2019 Arm Limited, all rights reserved

     Configuration
     =============

     Hardware: Mali-G76 r0p0
     Driver: Bifrost r19p0-00rel0
     Shader type: OpenGL ES Fragment

     Main shader
     ===========

     Work registers: 32
   ```

```
Uniform registers: 34
Stack spilling: False

                             A   LS    V    T  Bound
Total Instruction Cycles:  4.5  0.0  0.2  2.5      A
Shortest Path Cycles:      1.0  0.0  0.2  2.5      T
Longest Path Cycles:       4.5  0.0  0.2  2.5      A

A = Arithmetic, LS = Load/Store, V = Varying, T = Texture

Shader properties
=================

Uniform computation: False
```

2. Analyze the report. To decide which part of your shader code you need to optimize, identify the critical path units from the hardware units running in parallel. The performance table for the Main shader provides an approximate cycle cost breakdown for the major functional units in the design. For this shader you can see that:
    - The shader is texture bound when not using tone mapping. T is the highest value for the shortest path, taking 0.5 cycles a sample for this 5-sample blur. This is as fast as the hardware texture filtering unit in a Mali-G76 can go.
    - The shader is arithmetic bound when using matrix-based tone mapping. A is the highest value for the longest path when the conditional tone mapping block is executed.

For full details of all of the reported sections and fields, refer to the Mali Offline Compiler User Guide.

4. Optimize your shader

Now you have identified the critical path, speed up the tone mapping to improve performance of the shader.

1. The first change you can make is to reduce precision. Currently the tone mapping is using a highp (fp32) matrix operation, which has more precision than we need to generate an 8-bit per channel color output. Change the precision to “mediump” (fp16) float and sampler precision by modifying these two lines at the top of the shader:

   ```
   precision mediump float;
   precision mediump sampler2D;
   ```

   Just these two simple changes significantly reduce the cost of the longest path, as Mali GPUs can process twice as many fp16 operations per clock than fp32 operations.

   ```
                  A   LS    V    T   Bound
   Longest Path Cycles: 2.7  0.0  0.2  2.5       A
   ```

2. After changing the precision, arithmetic is still the longest path. Move the tone mapping out of the accumulation loop, and apply it to the final color instead of the individual samples. This gives the final shader structure:

   ```
   // For each gaussian sample
   for (int i = 0; i < WINDOW_SIZE; i++) {
    vec2 offsetTexCoord = texCoord + vec2(gaussOffsets[i], 0.0);
    vec4 data = texture(texUnit, offsetTexCoord);
    fragColor += data * gaussWeights[i];
   }

   // Tone map the final color
   if (toneMap) {
    fragColor *= colorModulation;
   }
   ```

   This change reduces the arithmetic cost of the longest path to just a single shader core cycle, even if tone mapping is used. The slowest path is now texturing, which needs 2.5 cycles per fragment to load the 5 samples needed. You can not make this any faster, because this is the architectural performance of this particular shader core.

   ```
                         A   LS    V    T   Bound
   Total Instruction Cycles: 1.0  0.0  0.2  2.5       T
   Shortest Path Cycles:     0.5  0.0  0.2  2.5       T
   Longest Path Cycles:      1.0  0.0  0.2  2.5       T
   ```

Although the last optimization reduced the arithmetic cost from 2.7 cycles to 1.0 cycles, the shader throughput only improved from 2.7 cycles to 2.5 cycles per fragment because the bottleneck changed from A to T. However, reducing the load on any pipeline will improve energy efficiency and prolong battery life, so these types of optimizations are still worth making, even if they do not improve the headline performance.

5. Target-aware profiling

Different models of Mali GPU are tuned for different target markets, so they have different performance ratios between the functional units. Mali Offline Compiler enables you to test the performance of your shaders on different target GPUs.

For example, compiling the shader for a Mali-G31, which is designed for embedded use cases, and therefore has a lower ratio of arithmetic to texture performance, returns the following performance report:

```
                           A   LS    V    T   Bound
Total Instruction Cycles: 2.9  0.0  0.2  2.5      A
Shortest Path Cycles:     1.6  0.0  0.2  2.5      T
Longest Path Cycles:      2.9  0.0  0.2  2.5      A
```

For this GPU, the shader is still arithmetic bound, even with the optimizations applied.

6. Limitations

The performance reports generated by Mali Offline Compiler are based only on the shader source code visible to the compiler. They are not aware of the actual uniform values or texture sampler configuration for any specific draw call, or any data-centric effects such as cache miss overheads.

The texture format and the filtering type used, can impact texture unit performance. Trilinear filtering (`GL_LINEAR_MIPMAP_LINEAR`) takes twice as long as bilinear filtering (`GL_LINEAR` or `GL_LINEAR_MIPMAP_NEAREST`), and anisotropic filtering can be scaled by both the probe type and the number of anisotropic sample probes made. Mali Offline Compiler assumes simple bilinear filtering for all samples, which is the fastest type supported by the hardware. If you know a draw call is using trilinear filtering for texture samples, you should double the cycle cost of the texture accesses reported in the performance report.

Arm Streamline, also included with Arm Mobile Studio, samples performance data from the Mali GPU hardware while your application runs on your target device. You can supplement Mali Offline Compiler performance reports with this data. For example, measure the number of multi-cycle texture operations being performed to validate the assumption that all accesses in your application are bilinear accesses.

7. Next steps

Explore your application further using more Arm Mobile Studio tools:

- Get started with Streamline
- Get started with Graphics Analyzer