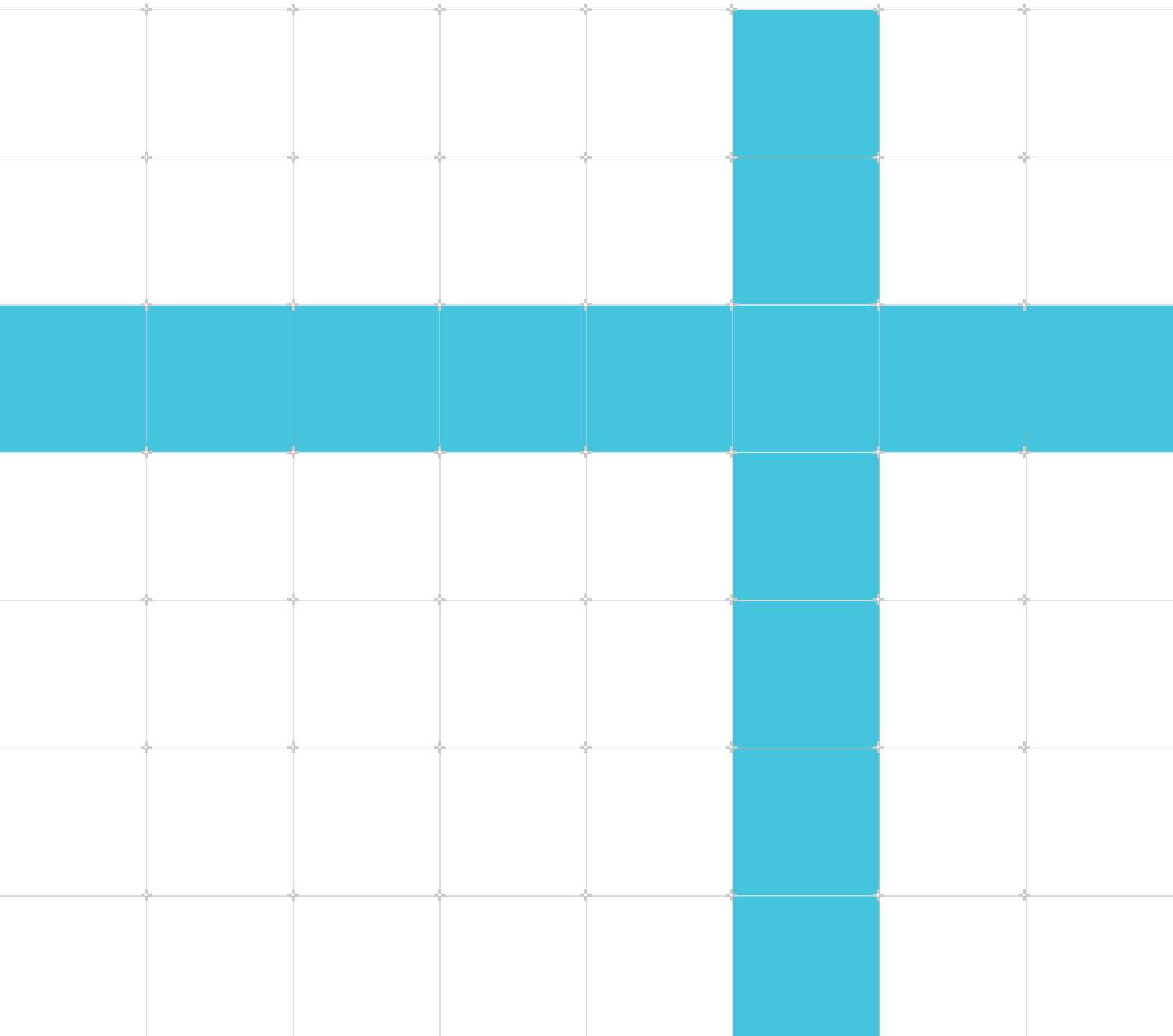
arm

Optimizing your software

Version 1.0

Issue 02
102442_0100_02_en



Optimizing your software

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	19 February 2019	Non-Confidential	First release

Proprietary Notice

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.

Arm Limited. Company 02557590 registered in England.

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.

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

Optimization means different things to different people. In some situations, you might simply want your code to run as fast as possible. However, if you are writing code for an embedded system, you might prefer to optimize for code density to reduce the memory footprint of your application.

Often these optimization constraints work against each other. For example, loop-unrolling is an optimization technique that can improve performance, but this optimization comes at the expense of increased code size. The first step in optimization is deciding what you want to optimize.

2. Performance analysis tools

A common phrase in software optimization is “you can only optimize what you can measure”.

This means that to improve the performance of your code, you must be able to accurately measure performance so that you can analyze where bottlenecks are occurring, make optimizations, and measure the improvements.

The right tool for this depends on the Arm platform you are working with.

RTOS and Android

Arm Streamline is the tool of choice for optimizing applications and systems for these platforms. Streamline is part of the Arm Development Studio and Arm Mobile Studio products:

- Using Streamline to Optimize Applications for Mali GPUs shows you how to use the Arm Streamline performance analyzer to optimize the graphics in Android applications running on Mali-400 and Mali T-600 GPUs.
- Streamline with the Raspberry Pi 3 is a practical tutorial that shows how to use the Arm DS-5 Streamline performance analyzer to analyze code performance running on the Raspberry Pi.
- Using Streamline with Fast Models and Fixed Virtual Platforms shows how you can analyze performance even when you don’t have a hardware target by using the Arm Fast Models and Fixed Virtual Platforms.
- Analyzing the performance of RTOS-based systems using Streamline shows how to analyze the performance of RTOS systems, using Keil RTX version 5 RTOS on an Arm Cortex-M33 processor.
- Streamline performance analysis using local capture mode shows how you can analyze performance when it’s not possible to send live data back to the host over a network or USB connection.

Linux

For Linux systems, there is a rich ecosystem of tooling from Arm and the open source community:

- For 64-bit Linux, Arm Forge MAP enables profiling of multi-threaded and multi-process C, C ++, Fortran and Python applications. Available as a standalone tool or as part of Arm Allinea Studio, Forge supports all major Linux distributions. The tool has advanced support for both CPU, application I/O, and MPI communication (for HPC applications). It can be used to identify end-to-end application performance problems such CPU bottlenecks, thread synchronization, and I/O problems.
    - Read about using Arm Forge to profile and optimize large scale high-performance computing (HPC) applications.
    - Find out more about Python profiling in our blog Profiling Python and compiled code with Arm Forge - and a performance surprise.
    - More information about Arm Forge MAP and the Arm Allinea Studio suite.
- Arm Streamline is a system profiler that can discover your software hot spots, via program counter sampling, as well as performance counter and process statistics. It can display per-core

and per-process hardware event counters from Arm CPUs and Mali GPUs. Streamline for Linux is available as part of Arm Development Studio.

- Perftools, the widely used command line Linux tools for hardware counters and other performance measurement, are available for Arm systems.

3. Coding best practices

How you write your source code can affect the efficiency of the executable code produced by the compiler. For example, loop counters that decrement to zero are generally more efficient than loop counters that increment to an arbitrary value. This is because the compiler can use a single instruction `SUBS` to decrement and compare to zero. Writing code that is more efficient delivers not only higher levels of performance, but can also be crucial in conserving battery life. If you can get a job done faster, in fewer cycles, you can turn off the power for longer periods:

- Coding considerations describes programming practices and techniques to increase the portability, efficiency and robustness of your C and C++ source code.
- Writing optimized code shows how you can use various options, pragmas, attributes, and coding techniques to make best use of the optimization capabilities of Arm Compiler.
- Using inline assembly to improve code efficiency is a tutorial that shows how you can write optimized assembly language routines to improve performance.
- Chapter 17 of the Cortex-A Programmer's Guide discusses how to optimize code to run on Arm processors.

4. Compiler optimization

The compiler provides many different options for optimizing the code it produces. For example:

- Vectorization enables the use of the Neon Single Instruction Multiple Data (SIMD) instructions that allow parallel processing of data.
- Link Time Optimization (LTO) increases the number of optimization opportunities by analyzing source code from different modules together.
- Function inlining can improve performance by reducing the overhead of repeated function calls.

These optimization techniques can be individually controlled using options supplied to the compiler and linker:

- Selecting optimization options shows how to select different optimization levels with Arm Compiler. For example, you can optimize for maximum performance, or for best code size.
- The armclang Reference Guide and armlink User Guide provide detailed descriptions of all available optimization options.
- Optimization Techniques in the Arm Compiler User Guide describes how to use armclang to optimize for either code size or performance, and the impact of the optimization level when debugging.
- Linker Optimization Features in the armlink User Guide describes the optimization features available in the Arm linker, armlink.