



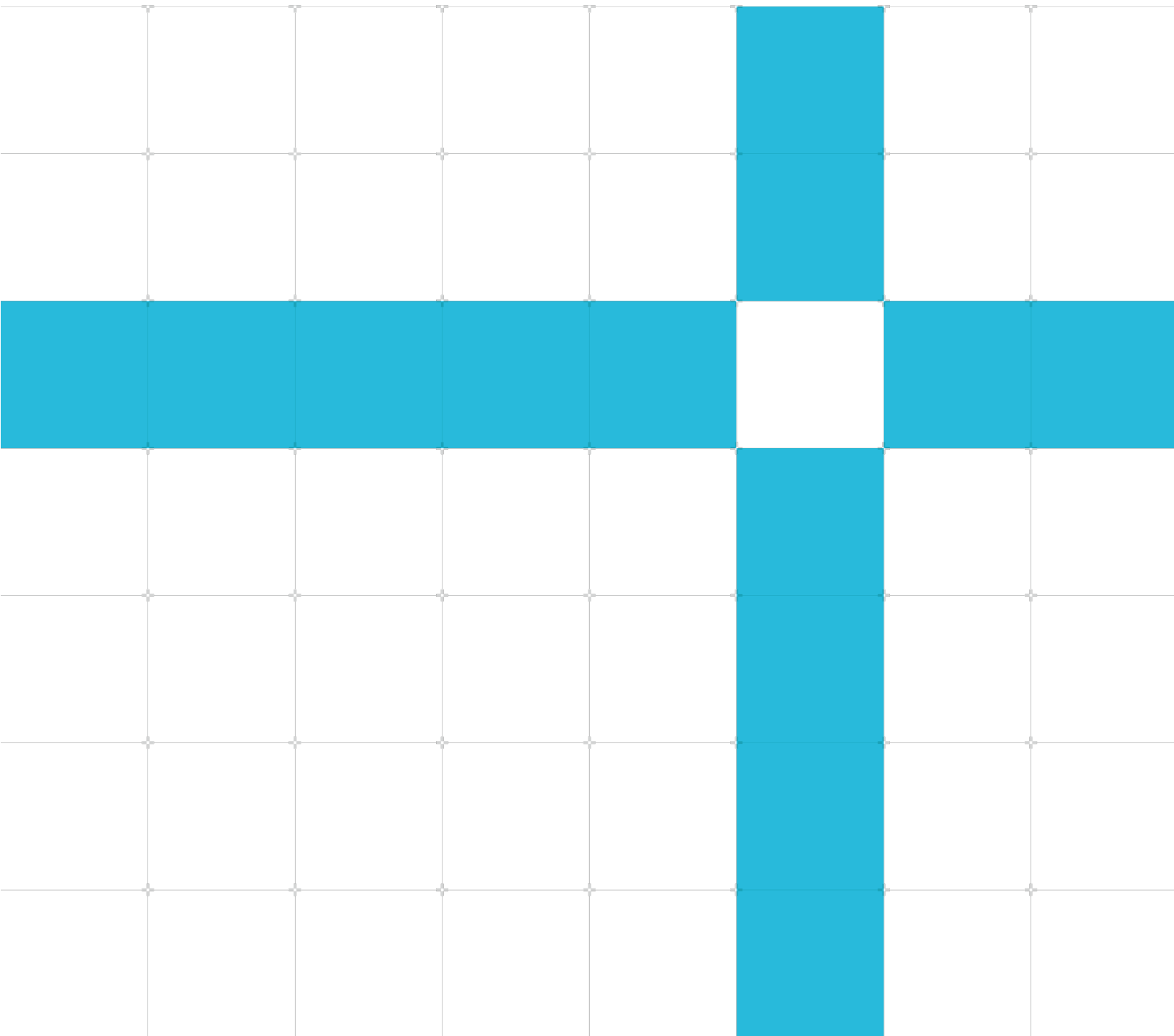
Workload Trace Generation

Revision: 0.0

Best Practices

Non-Confidential
Copyright © 2023 Arm Limited (or its affiliates).
All rights reserved.

Issue 01
107983



Workload Trace Generation

Best Practices

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
01	27-07-2023	Non-Confidential	First release

Non-Confidential 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, 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>.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.

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 Workload Trace Generation, 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

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

Contents

1. Introduction	5
1.1. Intended audience	5
1.2. Conventions.....	5
1.3. Useful resources	6
2. Defining a good trace.....	8
2.1. Why create a trace?	8
2.2. Trace components	9
2.2.1. Trace format	9
2.2.2. Trace restrictions.....	9
2.2.3. Trace size	9
2.3. Information in a trace	10
2.3.1. Focus on workload instructions.....	10
2.3.2. Limited context switching.....	10
2.3.3. Memory layout.....	10
3. Collecting a good trace.....	11
3.1. Run the workload on a reference system	11
3.1.1. Workload type	11
3.1.2. Reference workload performance	11
3.2. Run the workload on a virtual platform	12
3.2.1. Virtual platform selection	12
3.2.2. Tune the virtual platform	13
3.2.3. Turn on instruction tracing.....	13
3.2.4. PID/TID mapping.....	14
3.3. Verify the instruction trace	14
4. Conclusion	15
Appendix A. Revisions.....	16

1. Introduction

1.1. Intended audience

The intended audience for this document is a performance engineer preparing traces for performance prediction on new Arm hardware. It is expected that the reader understands the basic concepts of performance engineering such as:

- Workload characterization
- Identifying key aspects of a workload
- Understanding how Performance Monitor Unit (PMU) events correlate to a workload

1.2. Conventions

The following subsections describe conventions used in Arm documents.

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: <https://developer.arm.com/glossary>.







This document uses the following terms and abbreviations:

Terms and abbreviations

Term	Meaning
PMU	Performance Monitor Unit. A hardware counter measured while running a workload.
EL	Exception Level. One of four privilege levels defined by Arm specifications. Values range from EL0 to EL3.
PID	Process Identification number from the Linux kernel.
TID	Thread Identification number from the Linux kernel.
ASTF	Architectural Structured Trace Format. A standard developed by Arm for instruction traces.
TLB	Translation Lookaside Buffer.
TLBI	Translation Lookaside Buffer Invalidation.

Typographical conventions

Convention	Use
<i>italic</i>	Citations.

Convention	Use
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
<code>monospace</code>	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
<code>monospace</code> bold	Language keywords when used outside example code.
<code>monospace</code> <u>underline</u>	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: <code>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></code>
SMALL CAPITALS	Terms that have specific technical meanings as defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
 Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
 Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
 Danger	Requirements for the system. Not following these requirements will result in system failure or damage.
 Note	An important piece of information that needs your attention.
 Tip	A useful tip that might make it easier, better, or faster to perform a task.
 Remember	A reminder of something important that relates to the information you are reading.

1.3. Useful resources

This document contains information that is specific to this product. See the following resources for other relevant information.

- Arm Non-Confidential documents are available on developer.arm.com/documentation. Each document link in the tables below provides direct access to the online version of the document.
- Arm Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Fast Models Reference Guide	100964	Non-Confidential



Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.
Adobe PDF reader products can be downloaded at <http://www.adobe.com>.

2. Defining a good trace

2.1. Why create a trace?

Arm uses performance simulators to evaluate products in development. Feeding these simulators with relevant workloads helps in understanding how the product performs. A good workload for a performance simulator has well-understood characteristics that enable the simulation to analyze the effect of a design tradeoff.

Microarchitectural performance simulators are complex software programs to analyze design tradeoffs. Performance simulators run the instructions from a workload to compute the microarchitecture's performance behaviors for processing those instructions. The simulators are written for configurability and model detail, not for simulation speed, so the time required by the simulator to consume a workload limits the throughput of design tradeoff evaluations.

In general, it is best to share the full workload with Arm for more accurate hardware performance prediction. While these simulators might be able to run some binary workloads in execution-driven mode, they have some drawbacks:

- They might not include models of system devices, particularly if Arm did not develop the IP.
- They might not have the support software necessary to simulate the full execution of complex workloads.
- Some workloads require a lengthy instruction count to reach a point of interest, so the simulation of the setup is uninformative overhead.

If it is not possible to share full binaries and data for Arm to run, for example, for complexity reasons, an instruction trace may be a more concise workload representation. A trace contains the sequence of instructions executed architecturally, along with associated metadata for some instruction types. A simulator takes the instructions from a trace and computes the microarchitecture performance behavior for processing those instructions. An instruction trace is the preferred format when partners have reduced the workload to capture only the area of interest as it greatly reduces simulation time.

Capturing a good workload requires diligence. The workload functionality must be well-understood and its relevance to microarchitectural performance must be credible. The software first must go through source-level code optimization, compiled using an appropriately high level of compiler optimization or runtime just-in-time (JIT) compiler optimization. A good trace is captured from a properly performance-tuned workload configuration.

Creating a good trace is a complicated effort. The currently available tools require significant collaboration between the partner and Arm to create a useful trace. It is important that both parties are prepared to resource this effort. Given the resource requirements, the partner must allow Arm to include the instruction trace in its trace repository for regression analysis and future architectural improvements.

2.2. Trace components

2.2.1. Trace format

Arm developed the Architectural Structured Trace Format (ASTF) specification for instruction traces. Arm Fast Models support this format, and it is best suited for Arm's internal performance models.



Arm strongly recommends using the full ASTF specification for trace collection.

2.2.2. Trace restrictions

Traces are captured as one Architectural Structured Trace Format (ASTF) stream for each core. Traces should be collected at the system level for the best representation of the workload. EL2/EL3 interactions and multi-core simulations can be captured in an instruction trace but are not well represented in the current simulators from Arm. Expanding the simulation capabilities is under development.

The ASTF format currently supports only A-class AArch64. It does not support AArch32 or Morello security capabilities.

There are also some restrictions in its support for the AArch64 specification. It currently only supports v8.2-v8.5, without memory tagging extensions and range-TLBI. It supports SVE instructions, but the vector length cannot be changed in the instruction trace.

Virtualization, debug events, and Performance Monitor Unit (PMU) interaction should not be present in the simulation trace. Traces are targeted towards production-level workloads. Any debug or PMU instructions can produce unpredictable events in simulator behavior.

2.2.3. Trace size

A good trace is large enough to capture a workload's region of interest without extraneous information that increases simulation time. The ideal trace length is between 100 million and 10 billion instructions. Shorter traces do not have sufficient warm up for the simulation. Simulations for longer traces do not complete within a reasonable amount of time. Long traces can be sliced or sampled using simpoint methodology, however there is a much larger validation effort involved. Trace size reduction is best done by the workload experts.

Collecting the trace only during performance bottlenecks can help to keep the trace length reasonable. If there is repeated behavior in the area of interest, for example a compute-heavy loop, capture as few repetitions as necessary to replicate the loop behavior and memory access patterns.

2.3. Information in a trace

A good trace replicates in the simulator how the workload runs on real silicon. To achieve this, the trace must include certain components and exclude others. Sometimes this means tracing only one process out of many if only a single process is causing the performance bottleneck.

2.3.1. Focus on workload instructions

In a good trace, only execution level (EL) information relevant to the workload is collected, and non-workload related OS instructions are absent. In practice, this is generally not possible. The instruction trace should minimize the amount of OS and system interactions that are not relevant to the workload.

Trace capture of workloads involving significant OS, kernel, and device interactions is under investigation and no guidance is provided at this time. If the performance bottleneck is in a kernel function, tracing can help identify the issue, but user space tracing might not be relevant to increasing performance.

2.3.2. Limited context switching

The workload should be tuned so that single threads and processes run as continuously as possible. Any context switching should be relevant and representative of the rate observed on the silicon reference system. PID/TID maps are created during trace generation and processed with a script before trace consumption. If these maps are overly complicated by process creation or tear down during the trace collection, errors may be induced.

2.3.3. Memory layout

A good simulation from a trace mimics the real silicon environment as much as possible. As such, it is important to match memory layouts. Metadata containing information about page sizes and the mapping of large pages should be delivered with the instruction trace.

3. Collecting a good trace

3.1. Run the workload on a reference system

Before collecting a trace, it is important to understand how the workload runs on an existing reference silicon system. Arm usually does this by profiling the workload to find parts of interest and collecting performance metrics for those parts using the PMU. Any modern AArch64 system can be used, but the Arm® Neoverse™ N1 core has a well-established correlation to future silicon simulators and trace generation methods.



Profiling the workload on a reference system must be done for traces to be properly correlated.

3.1.1. Workload type

Understanding how the workload spends computational time on the reference system is important to collecting a good trace. If the workload has a steady state, then this area of the workload should be instrumented to begin collecting the trace. Manual toggling of trace capture and instruction stepping are other ways to ensure the relevant parts of the workload are collected in the trace.

A period of warmup instructions may also need to be collected so that the simulation is in the same state as the reference system when the period of interest is measured. An ideal method is to identify a series of consecutive iterations of a large loop whose performance is representative of subsequent iterations. If the performance varies significantly throughout the workload, separate traces should be captured. This allows analysis of why the variation occurred and how to optimize the overall performance.

3.1.2. Reference workload performance

Part of understanding a workload on a reference system is knowing how the workload performance is measured. A score or metric provided by the workload is the first stage of performance verification and workload tuning. Understanding any throughput metrics for the workload on the reference system is critical to determining if the traces are generating accurate predictions in simulators. It is critical that the workload performs as expected on the reference system before moving forward with collecting a trace.

In addition to any performance scores provided by the workload, system data is critical in matching virtual platforms to reference silicon. Ideally, PMU data for the workload is collected on the reference system and well-understood. A bare-metal reference system provides all the necessary PMU data. If

the reference system is in a cloud environment, PMU data collection might be restricted. The preferred reference system must be able to provide:

- Instruction count
- Core cycles
- L1/L2 cache misses
- L1/L2TLB misses
- Branch prediction statistics

By providing as much PMU data as possible, the traces in simulation can correlate to the reference system with a high degree of confidence, which ensures that the trace is a good representation of the workload. Multiplexing PMUs can affect data collection. So, when possible, use multiple runs to collect multiple PMU data.

In addition to performance metrics, in the Linux kernel configuration, `CONFIG_PID_IN_CONTEXTIDR` should be set to `Y` so when a context switch occurs, the PID in the kernel, or TID in user space, is stored in a hardware debug register. The virtual platform must have this configuration setting, so it is best to match that on the reference silicon.



Providing as much information about the reference system as possible, including operating system and memory configuration is critical to correlating traces in simulation to workloads on reference silicon.

3.2. Run the workload on a virtual platform

After a detailed understanding of the workload has been achieved on a reference system, you must run the workload on a virtual platform that matches the reference silicon as closely as possible. This helps to ensure the instruction trace is a good representation of the workload on silicon.

3.2.1. Virtual platform selection

Arm recommends using Fast Models with the ASTF plugin as the virtual platform for instruction trace tuning and collection. Arm provides many pre-built platforms as part of Fast Models. It also enables you to create your own platform using Fast Models tools. Reach out to your Arm contact if you require assistance customizing Fast Models to match your platform.

Sometimes, other virtual platforms or other tracing mechanisms may be appropriate. For example, if there are custom accelerators or some other non-Arm IP in the system, a platform such as DynamoRIO may be more appropriate. Each choice of platform comes with its own set of limitations and effort required to properly collect an instruction trace. As with the reference silicon system, as much information as possible about the virtual platform should be communicated with the instruction trace.



Arm does not officially support any virtual platform other than Fast Models for trace collection.

3.2.2. Tune the virtual platform

Fast Models provides a variety of options to adjust the model parameters during a simulation. Refer to the Fast Models documentation at <https://developer.arm.com/documentation/100964/latest> for complete details.

It is extremely important that the virtual platform matches the reference silicon as closely as possible. This includes OS versions, software libraries, and workload environment. Depending on the virtual platform selection, it may be easier to work backwards. For example, determine what the virtual platform looks like and then replicate that in the reference silicon.



In the Linux kernel configuration, `CONFIG_PID_IN_CONTEXTIDR` should be set to `Y` so that, upon a context switch, the PID in the kernel, or TID in user space, is stored in a hardware debug register for the tracing software to then record in the trace as the active TID at ELO.

The PID/TID mapping is also collected on the virtual platform and recorded in the trace. Non-Linux systems must provide `CONTEXTIDR` information. ASTF and associated tools were developed to process Linux PID/TID maps so efforts should be made by non-Linux users to ensure compatibility.

The workload should run on the virtual platform and match the results generated on silicon as closely as possible. It is important to compare all the data collected from the real system. It is possible that the performance metric may be within acceptable error limits while other throughput metrics vary significantly. If that is the case, the virtual model must be tuned so that both the primary performance metrics and any other throughput metrics match as closely as possible. For Arm internal simulations, the goal is less than 20% difference between simulation performance and silicon performance.

Once again, it is critical to communicate the reference silicon parameters and PMU data with the instruction trace. When a trace is delivered to Arm, an initial effort is made to correlate a simulator to the reference silicon. After this data is collected and verified, future hardware simulations can be performed with a certain degree of confidence.

3.2.3. Turn on instruction tracing

After tuning the virtual platform to estimate the workload behavior on real silicon, the virtual platform is ready to collect a trace. It is important to first decide between two options:

- Take one long trace or sample parts of the workload to collect instruction traces using instrumentation.
- Manually toggle trace collection.

You can collect the entire workload in one trace and check the length. If it is larger than 1 billion instructions, consider creating checkpoints in the workload or sample areas of the workload to reduce the overall length of the trace. Slicing the trace with post-processing tools can also be used to decrease the length of individual traces.

3.2.4. PID/TID mapping

When collecting the instruction trace, it is important to collect process and thread information. One method is to run the following command before and after tracing to map process IDs/thread IDs to command names:

```
ps wwH -eo 'pid,tid,comm,cmd'
```

With this command, however, the processes created and then reaped during the execution of the workload are lost. Alternatively, the `fttrace` command could also be used to let the Linux kernel log the mapping.

This mapping is used in a post-processing step as the `CONTEXTDIR` does not supply the relationship between threads and processes.

3.3. Verify the instruction trace

Arm has several tools that evaluate instruction traces for potential issues:

- The `trprint` tool has several options for viewing the content of the trace. It provides information such as the number of instructions, exception level ratios (EL0/EL1), and context records.
- The `trcheck` tool validates the captured trace according to the ASTF specification. It may highlight unexpected issues in a trace or whether features not yet supported in the ASTF format are present.

Evaluating the quality of a trace based on this information is not complete as there are many additional factors. As previously stated, generally the trace should have between 100 million to 10 billion instructions, expected exception level (EL) ratios, and context switches expected from the workload on reference silicon. Reach out to your Arm contacts for more information on how these variables affect the trace quality.

4. Conclusion

Collecting a good instruction trace is a complicated task. It is often iterative in nature, such as when tuning the virtual platform to match the reference silicon. Each step of the process has the potential to introduce error but also can only be correlated within a certain tolerance.

In general, whoever collects and verifies the trace should have a very good understanding of the workload and its performance. The amount of engineering effort to collect and verify a trace should not be underestimated. The process for each workload will vary slightly. It is important to communicate as much as possible during the trace collection process to not only create better quality traces but also to manage expectations for instruction trace consumption.

Appendix A. Revisions

This appendix describes the technical changes between released issues of this document.

The first table is for the first release. Then, each table compares the new issue of the book with the last released issue of the book. Release numbers match the revision history in Release Information on page 2.

Table A-1: Issue 0000-01

Change	Location
First early access release	-