



Workload Trace Generation

Version: 0.0

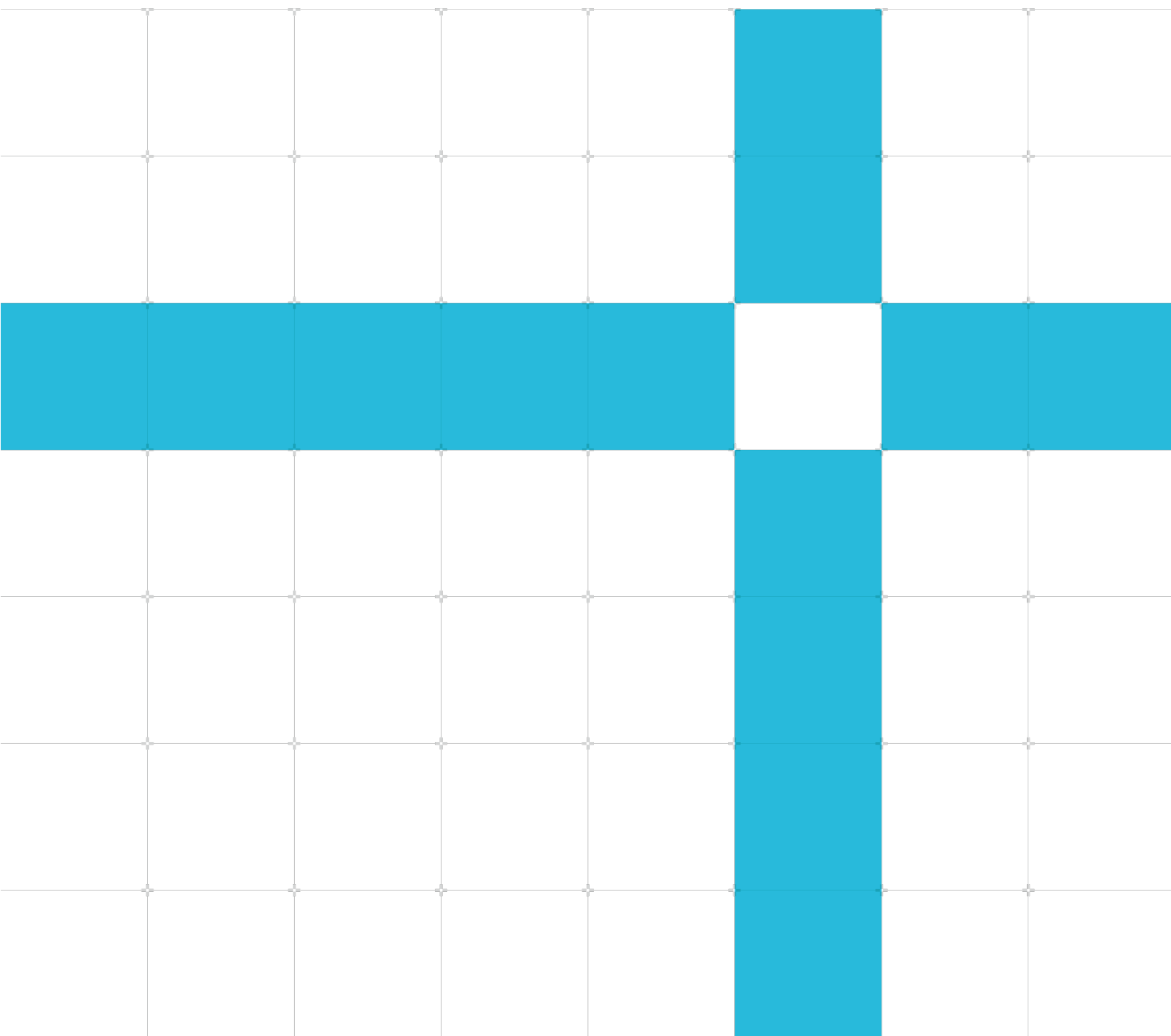
How to generate ASTF traces of workloads running on Fast Models

Non-Confidential

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

Issue 01

109193



Workload Trace Generation

How to generate ASTF traces of workloads running on Fast Models

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 for early access

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	6
1.1. Intended audience	6
1.2. Conventions.....	6
1.3. Useful resources	7
1. Preparing to collect a trace.....	9
1.1. Understand the Best Practices	9
1.2. Using Fast Models	9
1.3. Requirements checklist	9
1.4. System requirements	9
2. Workload preparation	10
2.1. Workload instrumentation.....	10
3. Running Fast Models for tracing	11
3.1. General Fast Model configuration.....	11
3.2. Tracing specific Fast Models configurations.....	11
3.2.1. ASTF tracing plugin	12
3.2.2. Toggling trace collection	12
3.2.3. Terminal connection for Fast Models (-S)	13
3.3. Checkpointing.....	13
3.3.1. Checkpointing with CRIU	13
3.3.2. Checkpointing with DMTCP.....	13
4. Example ASTF trace collection.....	14
4.1. Starting the FVP and connecting the debugger	14
4.2. Collecting a trace	16
4.3. Collect PID/TID information	20
5. Trace post-processing	21
5.1. Annotate the traces with PID/TID.....	21
5.2. Check the quality of the trace	21

Appendix A. Revisions..... 22

1. Introduction

1.1. Intended audience

This document is intended for performance engineers preparing traces for performance prediction on new Arm hardware. It is expected that you understand the basic concepts of performance engineering such as workload characterization, identifying key aspects of a workload, and understanding how PMU events correlate to a workload. You should have a basic knowledge of using Fast Models and you should have already read *Workload Trace Generation Best Practices*. It is assumed that you have retained those concepts and prepared a workload appropriately for capturing ASTF traces on Fast Models.

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>.

Typographical conventions

Convention	Use
<i>italic</i>	Citations.
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 bold</code>	Language keywords when used outside example code.
<code>monospace <u>underline</u></code>	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.

Convention	Use
 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 products	Document ID	Confidentiality
Fast Models Fixed Virtual Platform (FVP) Reference Guide	100966	Non-Confidential
Fast Models Reference Guide	100964	Non-Confidential
Fast Models User Guide	100965	Non-Confidential
Iris Python Debug Scripting User Guide	101421	Non-Confidential
Model Debugger for Fast Models User Guide	100968	Non-Confidential
Workload Trace Generation Best Practices	107983	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>.

1. Preparing to collect a trace

1.1. Understand the Best Practices

The Workload Trace Generation Best Practices document describes in detail what workloads are good candidates for tracing. It also explains what an effective trace captures and how large the trace should be. Read and understand this document before attempting to capture a trace.

1.2. Using Fast Models

Arm Fast Models are a suite of tools to help understand the Arm architecture and examine software behavior running in simulation. A good trace cannot be collected without a good understanding of how to use Fast Models and how the workload runs on real silicon. If possible, use a Fixed Virtual Platform (FVP) to run the workload in a Fast Models simulation. FVPs are created by Arm and represent the best adaptation of Arm Reference Designs in Fast Models. Running the workload on an FVP provides confidence that the methodology is correct before collecting data from a custom model. This guide uses FVP_Base_Neoverse-N1, which has 4 cores, and several Arm partners have silicon based on the Neoverse-N1 for direct comparison.

1.3. Requirements checklist

- Reference data from actual hardware as described in Workload Trace Generation Best Practices, for example PMU stats and workload output.
- A Linux disk image containing the workload.
- Firmware to boot the workload.
- A Fast Models platform, ideally an FVP provided by Arm such as the FVP_Base_Neoverse-N1.
- Model Debugger for interaction with and control of the FVP.
- ASTF plug-in, library, and tools (trprint, trcheck, trpidannotate, trimage, and trdd). Fast Models version 11.21 and later includes the plug-in by default.
- Telnet program such as PuTTY or a similar terminal emulator for connecting to and logging the FVP.

1.4. System requirements

Check the FVP system requirements for details as each FVP is compiled with a specific version of GCC. The Linux version must be used as ASTF is not currently supported on Windows. Fast Models can run on a local system, but it is recommended that you run it on a dedicated high-performance system for better performance. System RAM can be a limiting factor in Fast Models performance.

2. Workload preparation

As detailed in the Workload Trace Generation Best Practices documentation, the Fast Models environment and the reference hardware must match as closely as possible. The Linux image for Fast Models should be the same Linux distribution and major/minor version as on the reference hardware. Sometimes, it is easier to create the Linux image first and then boot the reference hardware with the same Linux version. In this guide, Ubuntu 20.04 is used.

The kernel option `CONTEXTIDR` should be enabled on both the Fast Models image and the reference hardware with the configuration option `CONFIG_PID_IN_CONTEXTIDR=y`. By default, most Linux versions including Ubuntu do not set `CONFIG_PID_IN_CONTEXTIDR`. The kernel must be re-built if the option is not correctly set as it can only be set when the kernel is built. This option allows the kernel to capture context information for the traces. Without it, the `PID` and `TID` are not added to the trace, and it is harder to confirm that the workload has been captured in the trace.

The regions of interest should already be identified in the workload as described in the *Best Practices*. Ideally, each region should comprise between 100 million and 10 billion instructions. If the region of interest is large and steady, tracing multiple samples across the region may be the most appropriate approach. It is important to decide before running Fast Models how the traces will be captured for the regions of interest. Only one method can be used during a single Fast Models run, either workload instrumentation or manual toggling. Manual toggling does not require any workload modification, but it requires a way to pause the model while it is running and toggle trace collection.

2.1. Workload instrumentation

The region of interest in a workload is demarcated by executing two `HLT` instructions, one each at the start and end of the region. An `HLT` instruction takes a 16-bit immediate value as an operand and generates an exception. How these `HLT` instructions are added to the workload depends on the programming language used to build the workload.

For instrumentation purposes, Fast Models can be configured to ignore `HLT` if its operand matches a pre-configured value. This value should be chosen carefully to be a value that is only used to toggle tracing. It is important that the region has an `HLT` instruction at both the beginning and end. The `HLT` instruction indicates a toggle for trace collection. If it is only present at the beginning of a region, then trace collection will not stop until the next `HLT` instruction. This will generate extraneous instructions in the trace and possibly cause the next region of interest to be skipped.

If multiple regions of a workload are being traced, a single run of the model may be able to capture all the traces. If each region is demarcated by executing two `HLT` instructions, then multiple traces sequentially numbered are captured. These traces represent each region of the workload executed between the two `HLT` instructions. The traces are numbered by the order in which they are collected during the execution.

3. Running Fast Models for tracing

Fast Models has a wide variety of configuration parameters. Since it is recommended that Fast Models run on a dedicated cluster, this chapter focuses on command-line options. These options can be enabled through a GUI if the Fast Model is being run locally or if X11 forwarding is enabled. Chapter 4 has examples from the GUI.

3.1. General Fast Model configuration

In addition to Fast Models general command-line options, every Fast Models FVP has a set of parameters as described in the Fast Models Reference Guide. The parameters are indicated by a `-C` flag.

Fast Models does not model timing events by default. To approximate timing in Fast Models, for example to produce a workload score, timing annotation can be enabled by setting the environment variable `FASTSIM_DISABLE_TA` to 0. When this variable is set, the proper Cycles Per Instruction (CPI) value can be set based on the reference hardware measurements. Use these two configuration options, which are common to all FVPs:

```
-C cluster0.cpi_div  
-C cluster0.cpi_mul
```

Example 4.1:

The reference hardware has a clock speed of 2.5 GHz, and the workload ran at an average of 1.25 Instructions Per Cycle (IPC). Therefore, the reference system executes 3125 million instructions per second. The parameters `cpi_mul` and `cpi_div` can be used to apply a fraction to each instruction. By setting `cpi_mul` to 1000, the `cpi_div` parameter reflects Millions of Instructions Per Second (MIPS). These parameters can only be set to integer values.

```
-C cluster0.cpi_div=3125  
-C cluster0.cpi_mul=1000
```

Several Fast Model runs may be necessary to tune the model to the correct CPI value from reference hardware. Other reference counters such as interrupt rate or a workload throughput can be used to tune the timing. Runtime configuration tuning is possible for CPI parameters as described in the *Iris Python Debug Scripting User Guide*.

3.2. Tracing specific Fast Models configurations

To tell Fast Models how to capture traces, you must set several command-line options specific to trace capturing. This is done by loading plugins into the FVP and then setting the parameters for the plugins. The order in which plugins are specified is important. The FVP loads each plugin in order of command line. This means if the workload has SVE instructions, the SVE plugin must be loaded first. Similarly, the ASTF plugin must be loaded before any plugins that may depend on it.

3.2.1. ASTF tracing plugin

Configure the ASTF plugin by first telling Fast Models where the plugin is located and then configuring the parameters for ASTF tracing. The `--plugin` flag points to the location of the ASTF plugin. Then several parameters such as compression method, trace file name, and timestamping of the traces can be controlled by the `-C TRACE.ASTFplugin` flags. For details of these flags, see the *Fast Models Reference Guide*.

Example 4.2:

The ASTF tracing plugin is located at `$PLUGINPATH`. The recommended compression to use for ASTF traces is Zlib as this provides a good balance between model performance and file size. The traces can be further compressed with post-processing tools. Each trace is prepended with `$FILENAME` and time stamping is enabled:

```
--plugin ${PLUGINPATH}/ASTFplugin.so \  
-C TRACE.ASTFplugin.encoding-method=2 \  
-C TRACE.ASTFplugin.trace-file=${FILENAME} \  
-C TRACE.ASTFplugin.timestamp-enable=1
```

3.2.2. Toggling trace collection

A second plugin is required so that trace collection can be dynamically toggled on and off to only trace the region of interest. First, set the plugin location, here it is `$PLUGINPATH`, and disable trace collection from the start of the simulation. If trace collection is not disabled from the start, all the Linux boot instructions and workload initialization are captured. This inflates the trace size unnecessarily and dramatically slows down the simulation:

```
--plugin ${PLUGINPATH}/ToggleMTIPlugin.so \  
-C TRACE.ToggleMTIPlugin.disable_mti_from_start=1
```

Next, set the method for toggling trace collection. If the workload was instrumented, Fast Models must know what HLT value to use to indicate the trace toggling. Here it is set to 1:

```
-C TRACE.ToggleMTIPlugin.hlt_imm16=0x1
```

Also, each CPU in the simulation must be configured to ignore the HLT value used for toggling traces. Here, there are 4 cores in the simulation and the trace toggling value is set to 1:

```
-C cluster0.cpu0.trace_special_hlt_imm16=1 \  
-C cluster0.cpu0.enable_trace_special_hlt_imm16=1 \  
-C cluster0.cpu1.trace_special_hlt_imm16=1 \  
-C cluster0.cpu1.enable_trace_special_hlt_imm16=1 \  
-C cluster0.cpu2.trace_special_hlt_imm16=1 \  
-C cluster0.cpu2.enable_trace_special_hlt_imm16=1 \  
-C cluster0.cpu3.trace_special_hlt_imm16=1 \  
-C cluster0.cpu3.enable_trace_special_hlt_imm16=1
```

To use manual toggling, set the value of `use_hlt` to 0:

```
-C TRACE.ToggleMTIPlugin.use_hlt=0
```

3.2.3. Terminal connection for Fast Models (-S)

The last configuration flag must be `-s`. This tells Fast Models to wait for a debugger to connect before starting. Since the order of flags is important, this flag is put last so that the FVP is properly set up before attaching the debugger. The parameters can be set after the debugger is attached, but that is not recommended.

3.3. Checkpointing

Fast Models does not natively support checkpointing. Two external, open-source tools are recommended for checkpointing with Fast Models: CRIU and DMTCP. Both methods have difficulties with portability. VirtIO disk image state is not handled by the checkpointing tools. The image state at the time of checkpointing must be manually copied along with the checkpoint files.

3.3.1. Checkpointing with CRIU

Checkpoint/Restore In Userspace (CRIU) is an open-source software project which can save and restore a Linux application container. Detailed information can be found on the [CRIU homepage](#). The general Fast Models parameters needed to use CRIU are:

```
-C bp.terminal_0.start_telnet=0  
-C bp.terminal_1.start_telnet=0  
-C bp.terminal_2.start_telnet=0  
-C bp.terminal_3.start_telnet=0  
-C bp.vis.disable_visualisation=1
```

3.3.2. Checkpointing with DMTCP

DMTCP is a wrapper that runs around Fast Models. To use DMTCP for checkpointing, the DMTCP software must be started before the Fast Model. DMTCP assumes fixed, absolute paths for binaries. Using DMTCP on a cluster with shared storage resolves the fixed path issue, but the DMTCP checkpoints do not run on different local machines. DMTCP also has a problem when Fast Models has an open telnet connection at the checkpoint. All telnet connections should be closed before taking a checkpoint. Consult the DMTCP documentation for more information.

4. Example ASTF trace collection

In this example, ASTF traces are collected from a 4 core FVP_Base_Neoverse-N1 on Fast Models 11.20. The workload is a version of SPECJBB tuned to a reference Graviton2 hardware run. The model is running on a dedicated node of a cluster. Xterm is used to interact with the model, but any terminal with X11 capabilities is acceptable. Be sure to start your ssh session with the `-X` parameter and to properly set up both the local and remote systems for X11 forwarding.

4.1. Starting the FVP and connecting the debugger

1. Connect to the machine on which the FVP will run, in this example 10.0.0.1:

```
ssh -X user@10.0.0.1
```

2. Launch the FVP. Here the FVP_Base_Neoverse-N1 is launched with parameters discussed earlier. `$PLUGINPATH` points to the ASTF tracing plugin location and `$PATH` includes the location of the FVP. Refer to the FVP documentation for a full description of all available options:

```
FASTSIM_DISABLE TA=0 FVP_Base_Neoverse-N1 \  
-C bp.secure_memory=0 \  
-C bp.dram_size=0x8 \  
-C bp.virtioblockdevice.image_path=specjbb_disk.raw \  
-C bp.terminal_0.terminal_command="xterm -title console_ttyAMA0 " \  
-C bp.pl011_uart0.unbuffered_output=1 \  
-C bp.terminal_1.terminal_command="xterm -title terminal_ttyAMA1 " \  
-C bp.pl011_uart1.unbuffered_output=1 \  
-C bp.terminal_2.terminal_command="xterm -title terminal_ttyAMA2 " \  
-C bp.pl011_uart2.unbuffered_output=1 \  
-C bp.terminal_3.terminal_command="xterm -title terminal_ttyAMA3 " \  
-C bp.pl011_uart3.unbuffered_output=1 \  
-C bp.vis.rate_limit-enable=0 \  
-C pctl.startup=0.*.*.* \  
--application cluster0.*=kiwi_specjbb_timerfix.elf \  
-C cluster0.cpi_div=${MIPS} \  
-C cluster0.cpi_mul=${DEFAULTMIPS} \  
--plugin ${PLUGINPATH}/ASTFplugin.so \  
-C TRACE.ASTFplugin.encoding-method=2 \  
-C TRACE.ASTFplugin.trace-file=specjbb_test \  
-C TRACE.ASTFplugin.timestamp-enable=1 \  
--plugin ${PLUGINPATH}/ToggleMTIPlugin.so \  
-C TRACE.ToggleMTIPlugin.disable_mti_from_start=1 \  
-C TRACE.ToggleMTIPlugin.use_hlt=0 \  
-S
```

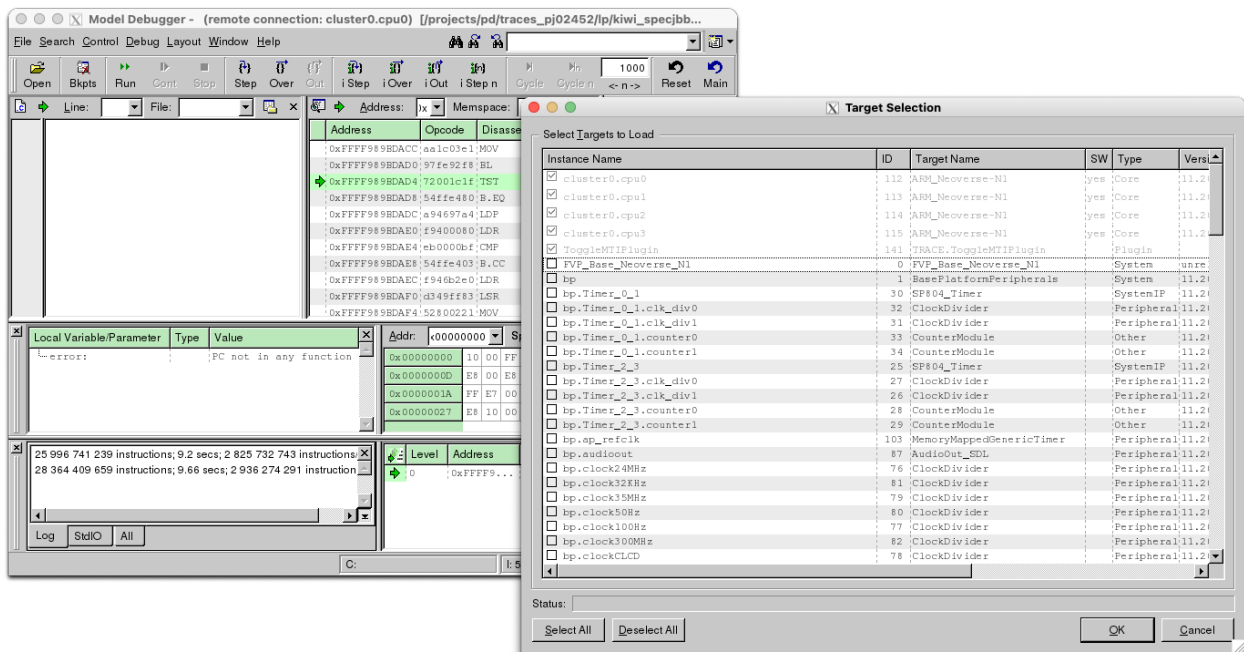
After starting, the screen should look like this:

```
Starting model...
terminal_1: Listening for serial connection on port 5000
terminal_2: Listening for serial connection on port 5001
terminal_3: Listening for serial connection on port 5002
terminal_0: Listening for serial connection on port 5003
ASTFplugin::addLoadStoreCallback trace source SVE_LOADS not detected -> omitting registration
ASTFplugin::addLoadStoreCallback trace source SVE_STORES not detected -> omitting registration
ASTFplugin::addLoadStoreCallback trace source SVE_LOADS not detected -> omitting registration
ASTFplugin::addLoadStoreCallback trace source SVE_STORES not detected -> omitting registration
ASTFplugin::addLoadStoreCallback trace source SVE_LOADS not detected -> omitting registration
ASTFplugin::addLoadStoreCallback trace source SVE_STORES not detected -> omitting registration
ASTFplugin::addLoadStoreCallback trace source SVE_LOADS not detected -> omitting registration
ASTFplugin::addLoadStoreCallback trace source SVE_STORES not detected -> omitting registration
Info: FVP_Base_Neoverse_N1: CADI Debug Server started for ARM Models...
```

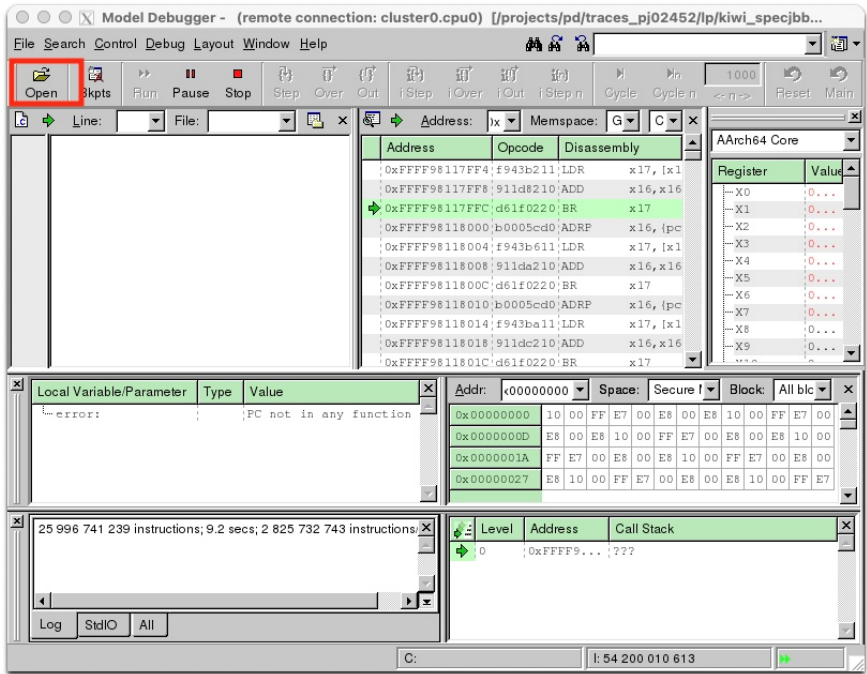
The model is now waiting for a debugger to connect and control the FVP.

3. Start a second ssh session and connect to the node running the model. From this session, initiate the `modeldebugger` and connect to the FVP through the **File > Connect to Model** option. The FVP is listed if the `-s` option was used and the `modeldebugger` is running on the same node.

When connecting to the model, all components of the FVP are available as debugger windows. Only select the necessary components, which are the cluster CPUs and the `ToggleMTIPlugin`. This allows pausing of the workload and toggling of trace collection.



4. When the debugger is connected, open the workload file on `cluster0.cpu0` to boot Linux in the model. This connects the GUI debugger to `cluster0.cpu0`:

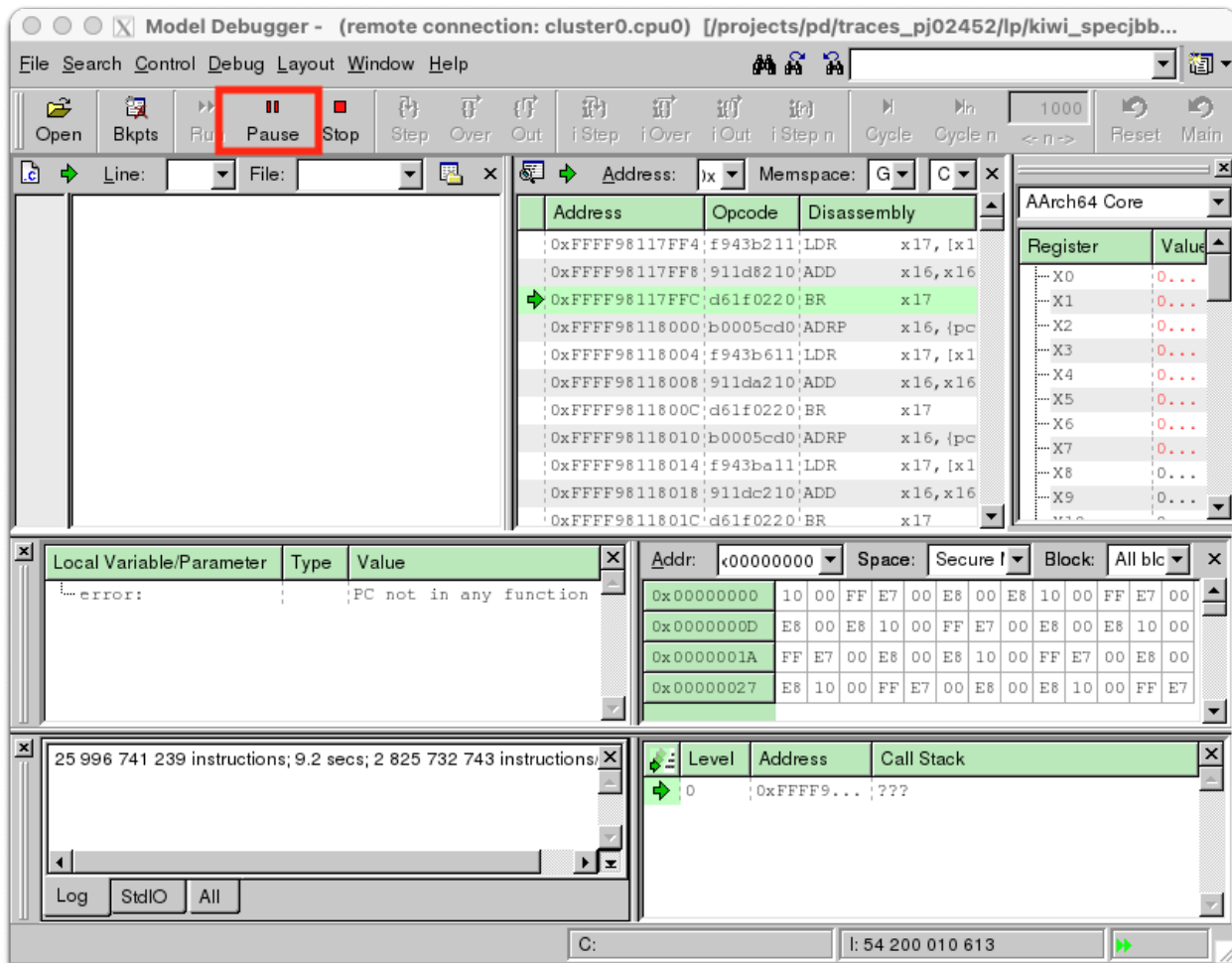


5. When the workload is opened, Linux should boot in a terminal window. After booting, start the workload as on the reference hardware.

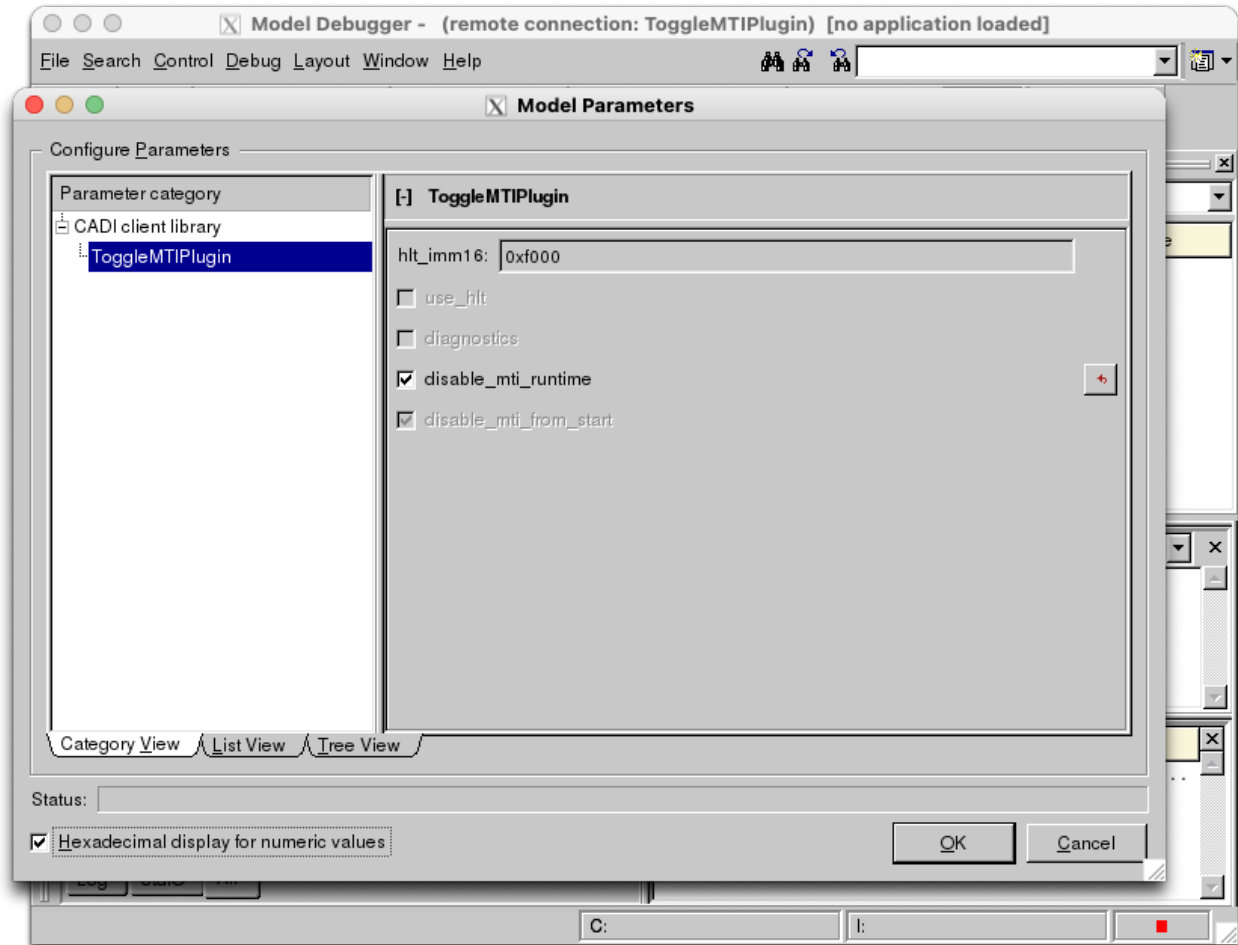
4.2. Collecting a trace

When the workload reaches the point of interest, the simulation is paused, and trace collection is enabled.

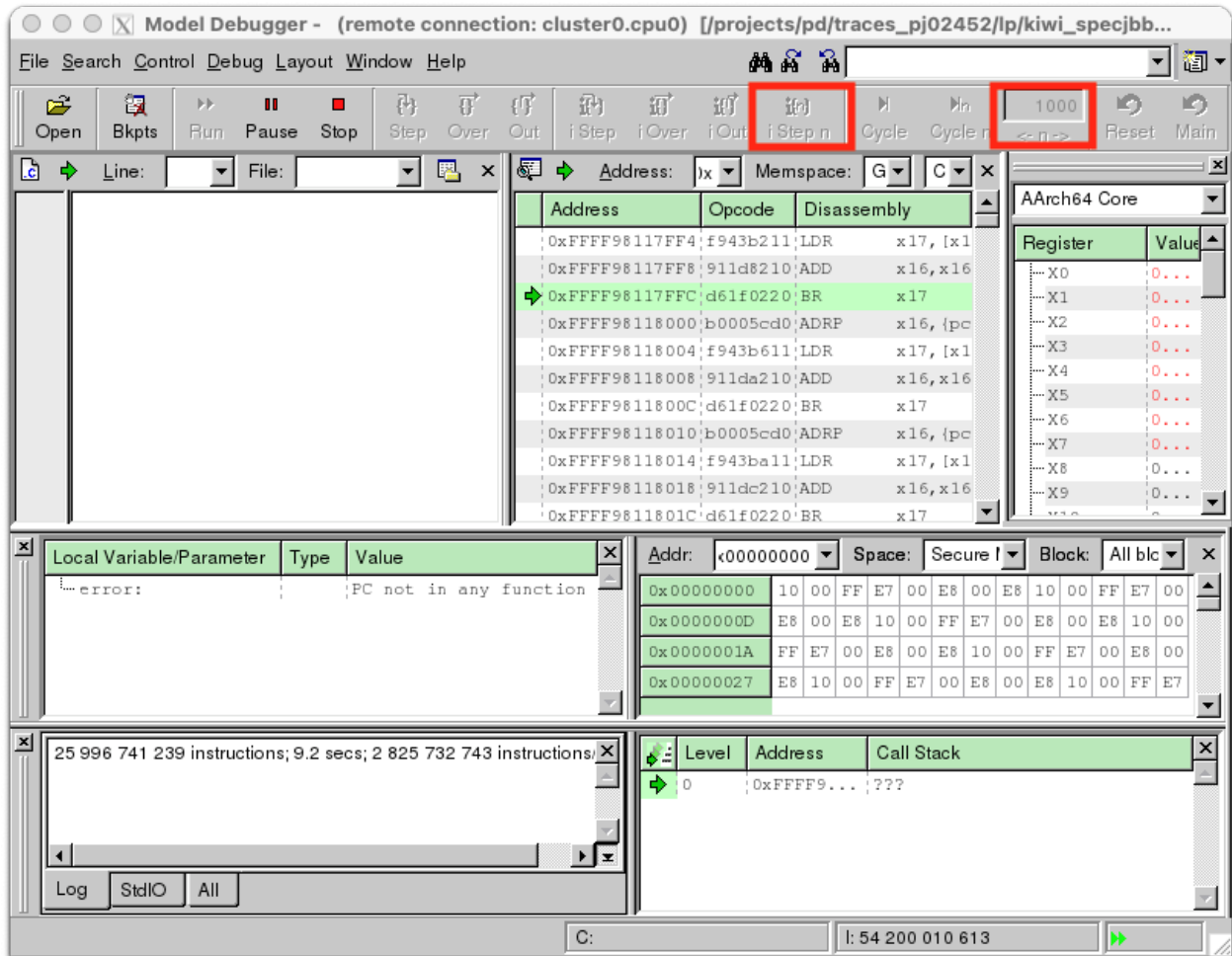
1. Pause the simulation in the `cluster0.cpu0` debugger window:



2. Move to the ToggleMTIPlugin window. Go to the **Debug** menu and select **Set Parameters**:



3. Select or clear the **disable_mti_runtime** checkbox to toggle trace collection. This box should be selected initially as trace collection was disabled on startup. Clearing the box and then resuming the simulation starts the trace collection.
4. If the region of interest is a known number of instructions, the debugger can be instructed to stop again after an instruction increment:



- After tracing is turned off, the trace files appear in the local directory from which the FVP was launched. The files are labeled based on the specified file name, cpu number, and trace sample number:

```
specjbb_test_FVP_Base_Neoverse_N1.cluster0.cpu0.0001.astf  specjbb_test_FVP_Base_Neoverse_N1.cluster0.cpu2.0001.astf
specjbb_test_FVP_Base_Neoverse_N1.cluster0.cpu1.0001.astf  specjbb_test_FVP_Base_Neoverse_N1.cluster0.cpu3.0001.astf
```

If tracing was not turned off and the simulation is terminated, .part is appended to the filenames to indicate they may not be complete:

```
specjbb_test_FVP_Base_Neoverse_N1.cluster0.cpu0.0002.astf.part
specjbb_test_FVP_Base_Neoverse_N1.cluster0.cpu1.0002.astf.part
specjbb_test_FVP_Base_Neoverse_N1.cluster0.cpu2.0002.astf.part
specjbb_test_FVP_Base_Neoverse_N1.cluster0.cpu3.0002.astf.part
```

The trace files are generated as trace collection is toggled off.

4.3. Collect PID/TID information

When the simulation has ended, collect the PID/TID information for trace annotation:

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

Save this output to a file called `pidtidmap`:

```
$ head pidtidmap
PID      TID  COMMAND                CMD
1         1  systemd                /sbin/init
2         2  kthreadd                [kthreadd]
4         4  kworker/0:0H            [kworker/0:0H]
6         6  mm_percpu_wq            [mm_percpu_wq]
7         7  ksoftirqd/0             [ksoftirqd/0]
8         8  rcu_sched                [rcu_sched]
9         9  rcu_bh                   [rcu_bh]
10        10  migration/0             [migration/0]
11        11  watchdog/0              [watchdog/0]
```

5. Trace post-processing

5.1. Annotate the traces with PID/TID

Use the `trpidannotate` tool to add PID/TID information to the trace. See the ASTF documentation for full details:

```
astf_tools/trpidannotate -i <input trace> -o <output trace> <pidtidmap file>
```

If the trace is not annotated with PID/TID information, the next step which checks the quality of the trace will fail.

5.2. Check the quality of the trace

Use the `trcheck` tool to determine if the trace meets ASTF standards. Refer to the ASTF documentation for detailed usage:

```
astf_tools/trcheck -P <trace_file>
```

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	-