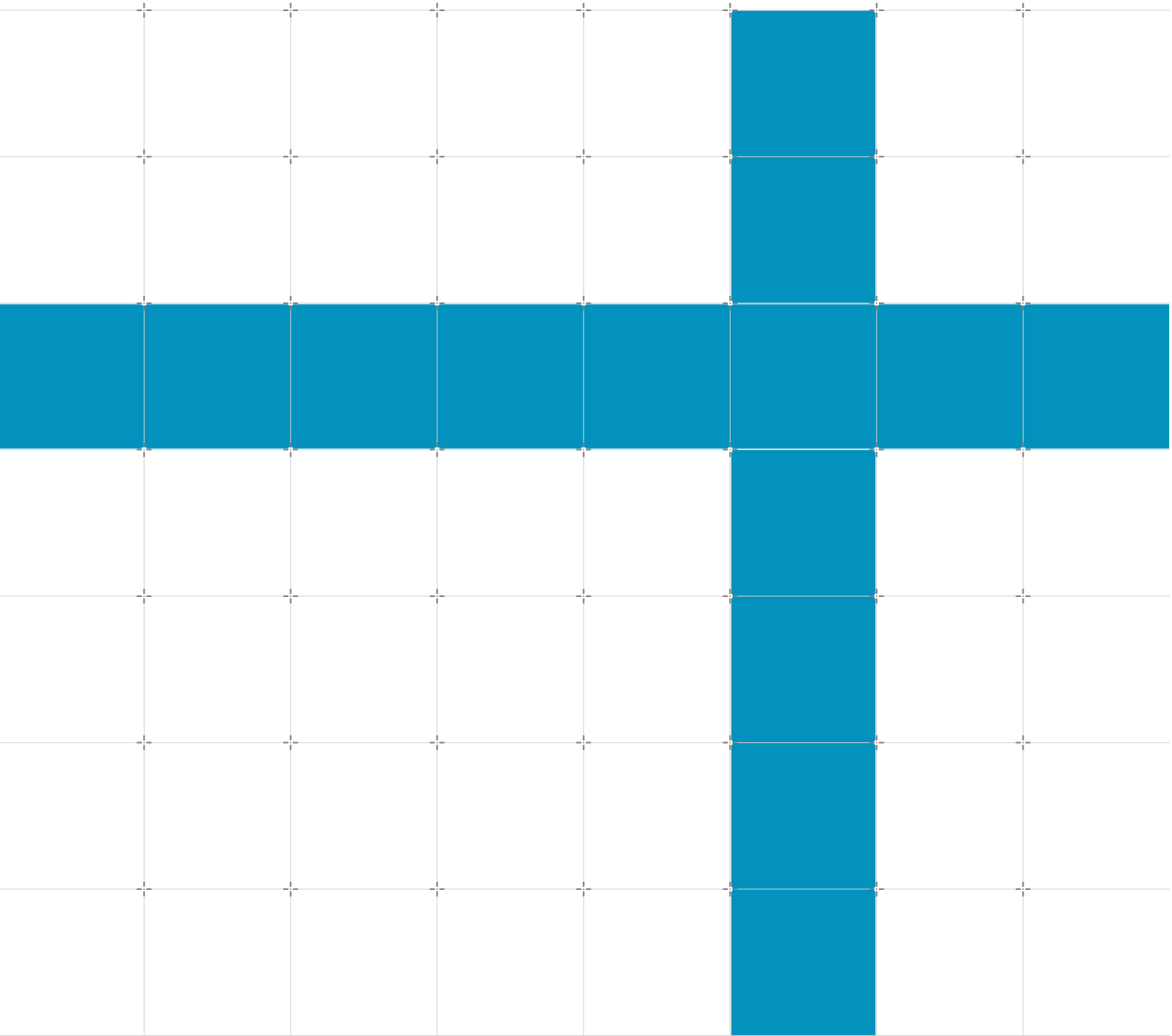
arm

Morello Instruction Emulator User Guide

Non-confidential

Issue

102270 0.3



Morello Instruction Emulator User Guide
Document ID: 102270

Release Information

Document History

Issue	Date	Confidentiality	Change
0.1	October 2020	Non-Confidential	Initial version Morello IE release 1.0.
0.2	February 2021	Non-Confidential	Clarification of known issues and limitations. New command line options. Updates of instruction and memory tracer. New tools: debugger and cache model. Morello IE release 1.1.
0.3	July 2021	Non-Confidential	Added description of new features. Updated known issue and limitations. Code examples now use Musl C library. Morello IE release 1.2.

Proprietary Notice

License

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, and 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.

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 for a Beta product, that is a product under development.

Web Address

https://www.arm.com

Contents

1 Introduction

1.1 Product revision status

Morello Instruction Emulator version: 1.2.

1.2 Intended audience

The Morello Instruction Emulator (Morello IE) is a tool for software developers and researchers who wish to experiment with the Morello architecture. It allows you to run userspace Morello applications on AArch64 Linux systems in a non-Morello environment. It also includes runtime instrumentation that can collect information about events and counters related to Morello. In addition, it includes an interactive debugger to help with running Morello applications.

The emulator can be used for:

- Experiments with Morello userspace applications on non-Morello AArch64 Linux systems.
- Evaluate compartmentalisation solutions and experiment with the Linux system call ABI.
- Test, debug and trace existing software being ported to Morello.
- Trace-based performance analysis and cache modelling for Morello applications.

Important: The Morello IE is an experimental tool and it does not provide a full Morello user space environment. Do not use the Morello IE to run code in a production environment.

1.3 System requirements

Morello IE is released in pre-built binary form and requires the following:

- Arm v8.0 hardware (v8.2 or above is recommended).
- Existing userspace GNU/Linux environment (for example, Red Hat 7.x or Ubuntu 18.04).
- The host system must have Glibc 2.17 or above.

1.4 Installation

The distribution bundle for Morello IE contains all required libraries and files. To install this tool, unpack the bundle at any location. To use the emulator launcher, add the `bin` directory of the installation folder to your `PATH` environment variable. The installation is self-consistent and portable.

```
$ tar -xf morelloie-${VERSION}.tar.gz
$ export PATH=${PATH}:$(pwd)/morelloie-${VERSION}/bin
$ morelloie -version
```

Morello architecture is backwards compatible with AArch64, and you can run an AArch64 application with the emulator. To check that your installation is successful, run the following command:

```
$ morelloie -- uname -m
```

This command runs `uname -m` in the emulator, and displays `aarch64`.

The installation directory includes:

- `bin` directory with launcher and frontend binaries.
- `lib` directory with instrumentation clients.
- `license_terms` directory with licence information.
- `README` a short readme file with a brief description of included items.

1.5 Related information

This document contains information that is specific to this product. See the following documents for other related information:

Reference	Document name	Document ID
[Morello IE]	This document	102270
[Morello AAPCS][1]	Morello extensions to PCS for the Arm 64-bit Architecture	102205
[Morello AAELF][2]	Morello extensions to ELF for the Arm 64-bit Architecture	102272
[Morello LLVM][3]	Morello extensions to ELF for the Arm 64-bit Architecture	102216
[CHERI][4]	CHERI C/C++ Programming Guide	

[1] https://github.com/ARM-software/abi-aa/blob/main/aapcs64-morello/aapcs64-morello.rst
[2] https://github.com/ARM-software/abi-aa/blob/main/aaelf64-morello/aaelf64-morello.rst
[3] https://git.morello-project.org/morello/llvm-project-releases/-/blob/morello/release-docs-1.1/UserGuide.pdf
[4] https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-947.pdf

2 Overview

2.1 General description

The Morello Instruction Emulator (Morello IE) is a dynamic binary translation tool. It is based on the JIT translation of each Morello instruction into a series of AArch64 instructions using DynamoRIO[5] instrumentation. The emulator maintains a consistent emulated CPU state as well as emulated memory tags. It also provides a layer of compatibility between Morello user space applications and non-Morello C library and kernel. This compatibility layer creates an environment which allows Morello applications to work on a non-Morello system.

The Morello IE can execute both hybrid and purecap Morello applications.

Two types of Morello applications are supported (see Fig 1):

- Applications linked to non-Morello C library (there are some limitations).
- Applications linked to Morello C library.

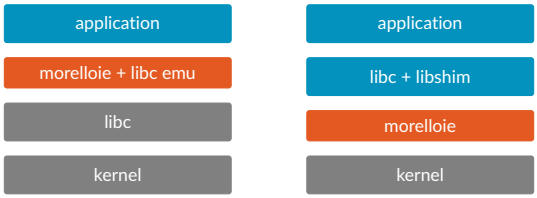


Fig. 1: Two types of supported purecap applications: with non-Morello C library (left) and Morello C library (right).

Components of the emulation layer for the C library API are automatically switched on or off based on the presence of purecap implementations of C library functions. You can modify this behavior with command line options. See Frontend options (page 9) for details.

The Morello IE implements the Morello ISA of version PROTO_REL_03.

2.2 Components

The Morello IE includes the following components:

- Emulator and debugger (`libmie.so` instrumentation client library).
- Tracer and cache model (`libtracer.so` instrumentation client library).
- Morello IE frontend (`morelloie` and `launcher` binaries in the `bin` folder).

[5] https://dynamorio.org/

Emulator

The Morello IE implements Morello instructions by replacing them with AArch64 code which can execute natively. The implementation relies on the emulated CPU state that is maintained by the emulator for each process thread. This state synchronizes on demand with real execution context, including the values of hardware registers and the objects and memory managed through the API of the C library.

Debugger

The interactive debugger gives access to the emulated state and runtime information for every executed instruction. It supports basic commands such as printing the emulated CPU state, working with PC-based breakpoints, and identifying location of the current execution point (for example, printing backtrace). The debugger provides access to data within the emulator that is not available by means of `lldb` or `gdb`. Both emulator and debugger are parts of `libmie.so` instrumentation client library.

Tracer

Instruction and memory access tracer is implemented as a separate instrumentation client for DynamoRIO and can be used independently of the instruction emulation client. For example, you can use it to analyze non-Morello AArch64 applications. It allows you to capture runtime trace of executed instructions and accesses to memory. You can configure the tracing scope to capture either the entire application execution flow, or a particular region(s) of interest.

Tracer also includes a module for collecting various micro-architectural statistics at runtime, including counters specific for Morello, and also data related to CPU cache modelling based on DynamoRIO's `drcachesim` tool. Tracer, statistics counter, and cache model are parts of the `libtracer.so` instrumentation client library.

Launcher

To run applications with instrumentation requires an additional launcher binary. This ''morelloie" binary loads emulation and tracer clients, and exposes application execution to the dynamic binary translation instrumentation. In addition, `morelloie` simplifies the use of Morello IE, and allows you to mange its various components from a single point.

An additional `launcher` binary is also available. You can use this to control which instrumentation client libraries are loaded. For example, this could be useful when you use a DynamoRIO instrumentation client along with the Morello emulation client.

Note: Arm recommends that you use the `morelloie` binary.

2.3 Command line options

This section gives an overview of all the command line options supported by the Morello Instruction Emulator (Morello IE).

Frontend options

Use the Morello IE through its frontend, the `morelloie` binary. The usage template is:

```
$ morelloie [options] -- application [application argument(s)]
```

The double-hyphen `--` acts as a separator. Specify options for `morelloie` before the `--`, and specify the options for your application after the `--`.

In general, you can invert any boolean option by adding `-no` in front of the option. For example, use `-no-enable-foo` to invert option `-enable-foo`. For options that have a value, separate the option and the value with a single space. For options that take a PC address `<pc>`, provide the address value as a hexadecimal literal with or without `0x` prefix. For example, `-break 21035c`. Specify options that accept integer values using decimal integer literals. For example, `-l1-d-size 1024`.

The frontend supports the following command line options.

Options to control instrumentation

`-v`
: Use verbose output from the launcher or frontend. The output is directed to `stderr`. This option is also accepted by the `launcher` binary.

  Default: verbose output is suppressed.

`-f`
: Use the `fork` syscall instead of `execve` to launch the process. This option is also accepted by the `launcher` binary.

  Default: `execve` is used to start new process.

`-fsz <n>`
: Specify maximum fragment size for instrumentation. This option limits the number of application instructions included in a single instrumented block of code, and sets the `max_bb_instrs` option of DynamoRIO. Increasing this value might improve instrumentation performance, but might also result in exceeding internal code cache limits in DynamoRIO. This option is accepted by the `launcher` binary.

  Default: `64`.

`-no-nmapi`
: Disable wrappers for non-Morello system API functions. By default, the Morello IE enables non-Morello API wrappers for C library functions that are compiled in A64 mode. If such a function is compiled in C64 (purecap) mode, the wrapper will be disabled automatically. Added in Morello IE 1.2.

  Use this option to override the automatic behaviour.

  This option also controls emulation for arguments of the `main` function.

Default: non-Morello API wrappers are selectively enabled based on the mode of startup code.

`-wrap-main`

Force non-Morello API wrapper for the `main` function. By default, wrapping of the `main` function is enabled for applications linked to a non-Morello C library and is disabled if the application has been linked to a Morello (purecap) version of the C library. Added in Morello IE 1.2.

Use this option to override the automatic behaviour.

Default: non-Morello wrapper for `main` is selectively enabled based on the mode of startup code.

`-capdesc`

Force processing of the `capdesc` relocation entries in the application (see [Morello AAELF]). By default, such relocation entries are processed for applications linked to a non-Morello C library and processing is disabled if the application has been linked to a Morello (purecap) version of the C library. Added in Morello IE 1.2.

Use this option to override the automatic behaviour.

Default: processing of the `capdesc` relocations is selectively enabled based on the mode of startup code.

`-skip-wrappers "name0,name1,name2,...,nameN"`

Skip certain wrappers for non-Morello system API. This option accepts a comma-separated list of function names. Added in Morello IE 1.2.

For example, use `-skip-wrappers "malloc,free"` to skip non-Morello API wrappers for the `malloc` and `free` functions only. Using `-skip-wrappers main` is equivalent to `-no-wrap-main`.

Use this option to override the automatic behaviour.

Default: empty string.

`-no-mie`

Disable use of Morello emulation client. This option can be helpful when only the tracer client needs to be used for analysing non-Morello applications. Only valid for `morelloie` binary.

Default: Morello emulation client is enabled.

Options to control Morello emulation client

Note: You can also supply the following options to the `libmie.so` instrumentation client. To do this, add the `-c path/to/libmie.so` option.

`-f-cpu`

Shows the CPU state on a capability fault. By default, this information is not shown when a capability fault occurs.

Default: false.

`-f-debug`

Enter debug mode on a capability fault. Normally, the application stops immediately after a capability fault. This option allows the client to enter debug mode for gathering additional information about runtime context (see Interactive debugger (page 24) for more information).

Default: false.

`-break <pc>`

Pauses execution immediately **before** executing the instruction at the given `<pc>` address and enters debug mode. If not specified explicitly, this option has no effect (there is no default value). Only a single instance of this option can be used on a command line.

Default: false. **Type:** address (accepts hexadecimal values).

`-debug`

Enable debug features. This option is automatically implied when using one of the `-f-debug` or `-break <pc>` options. If not specified explicitly, this option has no effect. This option must be used to enable tracking of instruction markers for the debugger (see Marker for debugger (page 15)).

Default: false.

`-no-strict-a64-store`

Disable tracking of tags for capabilities in memory during AArch64 stores. This stops invalidation of memory tags by AArch64 (non-Morello) store operations but also increases the speed of emulation.

Default: tracking of tags is enabled.

`-no-strict-c64-mem`

Disable checking all memory operations in purecap (C64) mode. This option ignores capability faults that would be generated by AArch64 (non-Morello) instructions but also increases the speed of emulation.

Default: all memory operations checking is enabled.

`-no-strict-pcc`

Disable PCC tag and permissions checking that normally happens at instruction fetch. Added in Morello IE 1.2.

Default: PCC permissions and bounds checking is enabled.

`-no-signal`

Do not emit OS signal on a capability fault. By default, every capability fault will result in a signal delivered to the application (see Capability faults (page 22)). This option can be used to override this behaviour. Added in Morello IE 1.2.

Default: signals are emitted for every capability fault.

`-verbose`

Show extra diagnostic messages from the Morello emulation client. When used with the frontend binary `morelloie`, this option affects both emulator and tracer.

Default: verbose output is suppressed.

`-verb-capdesc`

When used together with `-verbose`, this enables printing of the processed `capdesc` relocation entries. Added in Morello IE 1.2.

Default: verbose output is suppressed.

`-verb-libc-thunk`
When used together with `-verbose`, this enables printing additional information about wrappers for non-Morello C library API. Added in Morello IE 1.2.

Default: verbose output is suppressed.

Options to control tracer instrumentation client

Note: You can also supply the following options to the `libtracer.so` instrumentation client. To do this, add the `-c path/to/libtracer.so` option.

`-instr`
Enable instruction trace.

Default: instruction tracing is disabled.

`-mem`
Enable memory access trace.

Default: memory tracing is disabled.

`-trace`
Enable trace (instructions and memory), equivalent to using `-instr -mem` together.

Default: tracing is disabled.

`-stat`
Enable collection of micro-architectural statistics.

Default: collection of micro-architectural statistics is disabled.

`-cache`
Enable cache model and collecting associated data.

Default: cache model is not used.

`-full`
Deprecated in Morello IE 1.2, has no effect. Superseded by `-format` option.

`-format <csv,simple>`
Format for micro-architectural statistics and cache model results. Added in Morello IE 1.2.

Default: `simple`.

`-verbose`
Show extra diagnostic messages from the tracer instrumentation client. When used with the frontend binary `morelloie`, this option affects both emulator and tracer.

Default: verbose output is suppressed.

Options to control scope of tracing

By default, the scope of tracing corresponds to everything executed from the start of the `main` function up to and including the return from it. However, if you run an application without symbol information, all code is traced.

Use the following options to modify the scope tracing:

Note: The four following sets of options are mutually exclusive.

`-all`
: Trace all instructions that execute, this includes instructions that execute outside of the `main` function.

`-roi`
: Trace only instructions from the region(s) of interest which are defined by the tracer marker instructions (see Markers for tracing (page 15)).

`-c64`
: Trace only instructions executed in purecap (C64) mode.

`-fr <pc1> -to <pc2>`
: Define the tracing region of interest with addresses `<pc1>` and `<pc1>`. You must supply these two options together.

`-main <name>`
: Use the name `<name>` instead of `main` for default tracing scope. This can be useful when the actual `main` function consists of a single branch instruction. Added in Morello IE 1.2.

Options for cache model

You must add the `-cache` option to use any of the following options:

`-l1-d-size`
: L1 d-cache size (bytes), and size must be power of two.

  Default: 64 KiB. **Type:** integer (accepts integer decimal values).

`-l1-i-size`
: L1 i-cache size (bytes), and size must be power of two.

  Default: 64 KiB. **Type:** integer (accepts integer decimal values).

`-l2-size`
: L2 cache size (bytes), and size must be power of two.

  Default: 512 KiB. **Type:** integer (accepts integer decimal values).

`-cache-line-size`
: Cache line size (bytes), and size must be power of two.

  Default: 64 bytes. **Type:** integer (accepts integer decimal values).

`-l1-d-ways`
: L1 d-cache associativity, must be power of two.

  Default: 4. **Type:** integer (accepts integer decimal values).

`-l1-i-ways`
: L1 i-cache associativity, must be power of two.

  Default: 4. **Type:** integer (accepts integer decimal values).

`-l2-ways`
: L2 cache associativity, must be power of two.

  Default: 8. **Type:** integer (accepts integer decimal values).

`-cache-cores`
: The number of cores in the cache model.

  Default: 1. **Type:** integer (accepts integer decimal values).

`-cache-prefetcher <prefetcher>`
: Hardware data prefetcher policy: `nextline` or `none`. Added in Morello IE 1.2.

  Default: `none`.

`-cache-rep-policy <policy>`
: Cache replacement policy (`LRU` – least recently used, `LFU` least frequently used, `FIFO` – first in first out). Added in Morello IE 1.2.

  Default: `LRU`.

Miscellaneous options

`-help`
: Display command line options for the instrumentation clients.

`-h`
: Display command line options for the launcher executable.

`-version`
: Display version information and exit.

Note: When you use the `morelloie` frontend binary, if you do not specify tracer options, the tracer client will not load. This improves the performance of the emulation. To disable loading of Morello emulation client `libmie.so`, use the option `-no-mie`. This option is only available for the `morelloie` frontend.

Launcher options

Note: Arm recommends that you always use the `morelloie` frontend binary.

When the `launcher` binary is used directly, the usage template is:

```
launcher [options] [-c /path/to/lib/libtracer.so [tracer-options]] \
[-c /path/to/lib/libmie.so [emulator-options]] -- application [application argument(s)]
```

The general rule is that any options supplied before the `-c` option are consumed by the `launcher` itself. The `-c` option:

- starts command line for each instrumentation client,
- must be followed by the path to the client's library and optionally by the arguments intended for this client.

All options between `-c` and either the next `-c` option or the double hyphen delimiter `--` are part of the command line for the corresponding client library. One or more instrumentation clients can be used together. Everything after the `--` delimiter forms the command line to start the process of the application.

Most of the options described in Frontend options (page 9) also apply here.

2.4 Inline control instructions

The Morello IE supports special marker instructions that you can embed in the source code applications. The following section describes the macros you can use to control the tracing scope or to introduce breakpoints for the built-in debugger.

Markers for tracing

You can use the following macros to embed start and stop tracing markers. Every time the execution reaches these instructions, tracing is enabled or disabled. To use these markers, specify the `-roi` option. Without this option, marker instructions have no effect. For example:

```
/* Start tracing */
#define __START_TRACE() __asm__ volatile ("hint #0b1000000")

/* Stop tracing */
#define __STOP_TRACE()  __asm__ volatile ("hint #0b1000001")
```

The instructions that result from this code are valid AArch64 instruction equivalents of `NOP` and do not affect the functionality of the application.

Marker for debugger

When you enable debug mode, you can use the following macro to insert a breakpoint which the Morello emulation client will recognize. Use `-debug` option to enable this mode.

```
/* Put breakpoint */
#define __MIE_DEBUG() __asm__ volatile ("hint #0b1000100")
```

The instructions that result from this code are valid AArch64 equivalents of `NOP` and do not affect the functionality of the application.

2.5 Limitations and known issues

The Morello IE can run both purecap and hybrid Morello userspace applications on non-Morello AArch64 Linux systems. (See [Morello AAPCS for more details.) However, pay attention to the exceptions and limitations as described below.

Running dynamically linked applications

On systems without a Morello-aware dynamic linker, dynamically linked purecap Morello applications do not run because of the lack of support for Morello relocations. To workaround this, you can build statically-linked purecap applications that execute under the Morello IE.

Dynamically and statically linked hybrid Morello applications should work as expected.

Running stripped binaries

Stripped binaries that do not contain any symbol information may not run under the Morello IE, because of the lack of symbol names that are required for correct detection of C64 thunks. This is required for the C library emulation layer to function properly.

When linking a Morello application to a Morello-aware version of the C library, this type of application will work correctly and there is no limitation.

By default, in stripped applications the entire execution flow is traced. By default, in non-stripped applications, tracing is enabled only for the code invoked by the `main` function.

Use of a non-Morello C library

On systems with no Morello C library, you can still build and link hybrid and purecap Morello applications using the existing (non-Morello) C library. This method allows you to create Morello applications that you can execute under Morello IE that emulate Morello-aware interface for the application. This helps you to experiment with Morello without having to port a C library to Morello.

Note: Non-Morello C library functions execute in A64 mode, and do not have Morello functionality.

Important: See Linking to non-Morello Glibc (page 17) for more details.

Inexact bounds

For a large allocation using `malloc` or a similar function, the returned capability may have bounds greater than the size requested, because Morello capabilities cannot exactly represent every possible combination of address and bounds.

Objects with inexact bounds allocated by the non-Morello `malloc` family of functions have their capability bounds set correctly which means that inexact bounds apply. However, the size of the allocated memory is exactly as requested and the capability bounds might be wider than the allocated size. There is a risk that unrelated heap objects have overlapping bounds.

Use `-verbose` option to display a message whenever inexact capability bounds are used.

Linking to Glibc with ifuncs

On systems with no Morello dynamic linker, purecap applications linked statically to Glibc with ifuncs will not function properly. This is related to the lack of dynamic linker support for processing of relocation entries required for ifuncs to be loaded correctly. A workaround is to use another C library (for example, Musl libc[6]) instead of Glibc.

Linking to non-Morello Glibc

When linking to a non-Morello version of Glibc, note that Glibc startup code manages a number of objects in memory. These objects will have incorrect offsets if an object is used from C64 (purecap) code while being defined in A64 code. For this reason an application linked to a non-Morello Glibc will most likely crash with the `SIGSEGV` error. For this reason, if a non-Morello C library is required, Arm recommends that you use the Musl C library.

CCTLR.SBL

Effects of `CCTLR.SBL = 1` are not fully supported in C64 (purecap) mode. The modifications to the AArch64 instructions that are related to `CCTLR.SBL` control bit are not yet implemented. The Morello instructions that are affected by the `CCTLR.SBL` control bit value are implemented correctly.

Multi-threaded applications

Support for multi-threaded applications based on `libpthread` is incomplete.

[6] https://musl.libc.org/

3 Advanced topics

3.1 Building Morello applications

You can use the Morello LLVM toolchain to build Morello applications. See [Morello LLVM] for more details about using the toolchain. This section shows some examples of building and running such applications. The following examples use Musl C library for linking purecap Morello applications (both non-Morello and Morello[7] versions of this library can be used).

In the examples below, the **${MUSL}** variable refers to the installation directory of the Musl C library. The **${MORELLO}** variable refers to the installation folder of the Morello LLVM toolchain.

Note: The Morello LLVM toolchain can be built from source for AArch64 Linux host systems. Use the sources from https://git.morello-project.org/morello/llvm-project-releases and the commands described in Building Morello LLVM for AArch64 Linux hosts (page 30).

The current version of the emulator 1.2 can execute:

- statically linked purecap (C64) Morello applications,
- statically and dynamically linked hybrid Morello applications.

Note: See the [Morello AAPCS] for more details about the different types of Morello applications and their execution modes.

Running an application under the Morello IE generates normal application output to **stderr** and **stdout**, while the output of the emulator itself always redirects to **stderr**.

Build and run a simple application

In the following examples, the **clang** command refers to the C compiler from Morello LLVM toolchain.

The following example shows the simplest hello world example for a purecap Morello application:

```
// hello.c
#include <stdio.h>

int main() {
    printf("hello Morello\n");
    return 0;
}
```

You can use most standard C library headers installed in your system to write source code of Morello applications. However, you must patch some of the headers (such as **libio.h** from Glibc) to make them compatible with Morello.

You can work around this issue by not including headers that contain references to the files requiring modification (for example, **stdio.h** when using Glibc headers). Instead, you can declare the functions explicitly:

[7] https://git.morello-project.org/morello/musl-libc

```
// hello.c
int printf(const char *format, ...);
```

instead of

```
// hello.c
#include <stdio.h>
```

To compile the hello world example, use the following command:

```
$ clang -c -nostdinc -isystem ${MUSL}/include -march=morello+c64 -mabi=purecap hello.c -o hello.c.o
```

If the host system does not support Armv8.2 instructions, use `-march=armv8-a+c64` instead of `-march=morello+c64`. This instructs the compiler to only emit Arm v8.0 code. This should not affect Morello functionality, however wherever possible Arm recommends that you use `-march=morello+c64`.

To link this example in a purecap Morello application, use this command:

```
$ clang -fuse-ld=lld -Wl,--morello-c64-plt -o hello \
    ${MUSL}/lib/crt1.o ${MUSL}/lib/crti.o \
    $(clang -print-file-name=clang_rt.crtbegin-aarch64.o) \
    hello.c.o \
    $(clang -print-file-name=clang_rt.crtend-aarch64.o) \
    ${MUSL}/lib/crtn.o -nostdlib -L${MUSL}/lib -lc -static
```

To run this example, use:

```
$ morelloie -- ./hello
```

Alternatively, you can run an application under Morello IE by using the `launcher` binary explicitly:

```
$ launcher -c /path/to/lib/libmie.so -- ./hello
```

When targeting purecap (C64) applications, to enable Morello support provide the options:

- `-march=morello+c64` and `-mabi=purecap` for compilation;
- `-Wl,--morello-c64-plt` for linking.

For hybrid (A64) execution, to enable Morello support, specify the options:

- `-march=morello` for compilation;
- `-Wl,--morello-c64-plt` for linking.

To link to the Morello application binary, use the LLVM linker from the Morello toolchain providing `-fuse-ld=lld` to invoke LLVM linker. To indicate that the binary must be statically linked, use the `-static` option.

Stack corruption example

This example demonstrates a deliberate capability fault that results from out of bounds access to memory protected by a capability.

Out of bounds writes to memory, allocated on stack, can modify content that is referenced by another variable. This process is also known as stack corruption. The following example demonstrates how a Morello application behaves when stack corruption is about to happen.

```
// stack.c
void fun(int *data) {
    data[3] = 3; // <--- access outside object bounds
}
int main() {
    int x = 0;
    int data[3] = {0, 1, 2};
    fun(data);
    return data[0] + x;
}
```

Try running this example:

```
$ morelloie -- ./stack
Simulated capability fault: out of bounds access at 2342d4
Range [0000ffffdf2a81e8, +4] is not in bounds [0000ffffdf2a81dc, 0000ffffdf2a81e8]
Fault capability: 0x1:ffffc000:41e881dc:0000ffff:df2a81e8
         tag: true
       value: 0x 0000ffffdf2a81e8
              ...
       valid: true
   in bounds: false
      length: 12
      offset: 12
              ...
Segmentation fault (core dumped)
```

This interrupts the execution and delivers `SIGSEGV` to the application due to an out of bounds access capability fault.

```
2342d0: 68 00 80 52                  mov     w8, #3
2342d4: 08 0c 00 b9                  str     w8, [c0, #12] // <---
```

Note: To explore runtime context when an out-of-bounds fault occurs, run the command with the `-f-debug` option. This enters debug mode when and out-of-bounds fault occurs.

Out of bounds access example

The following example shows the behavior of a Morello application when an out of bounds access occurs for a dynamically allocated block of memory that is protected by a capability.

```
// heap.c
#include <stdlib.h>

int main() {
    int *data = (int *)malloc(sizeof(int) * 3);
    int x = data[3]; // <--- reading outside of bounds
    return x;
}
```

To run the example:

```
$ morelloie -- ./heap
Simulated capability fault: out of bounds access at 2342f8
Range [00000000002f0c8c, +4] is not in bounds [00000000002f0c80, 00000000002f0c8c]
Fault capability: 0x1:dc100000:4c8c0c80:00000000:002f0c8c
         tag: true
       value: 0x 00000000002f0c8c
              ...
       valid: true
   in bounds: false
              ...
Segmentation fault (core dumped)
```

Build and run hybrid Morello application

You can define capabilities explicitly in a hybrid Morello application. To do this, wrap a pointer into a capability to enable Morello to protect the memory access. For example:

```
// hybrid.c
#include <stdlib.h>

int main() {
    int* __capability cap = (__cheri_tocap int* __capability)malloc(3 * sizeof(int));
    __asm__ volatile ("hint #0b1000000"); // start tracing
    int x = cap[3]; // <--- reading outside of bounds
    __asm__ volatile ("hint #0b1000001"); // end tracing
    free((__cheri_fromcap void *)cap);
    return x;
}
```

To build this into a hybrid Morello application, run the following commands:

```
$ clang -march=morello -fuse-ld=lld hybrid.c -o hybrid
```

Try running the example with the `-instr` or `-trace` options together with `-roi` option, to enable instruction tracing for the region of interest. For example:

```
$ morelloie -trace -roi -- ./hybrid
M      1 21082c (A64): c24007e1 ldr    Ct=1 Rn=31 imm12=1 CapLoadImmPre
M        21082c (A64): MR16            0000fffff427e480 feffc000401c00100000ffff1a950010
M        21082c (A64): CR01            0000fffff427e480 1
M      2 210830 (A64): e280c428 ldur   Rt=8 Rn=1 imm9=12 AltLoadWordImmUnscaled
Simulated capability fault: out of bounds access at 210830
Range [0000ffff1a95001c, +4] is not in bounds [0000ffff1a950010, 0000ffff1a95001c]
Fault capability: 0x1:feffc000:401c0010:0000ffff:1a95001c
      tag: true
    value: 0x 0000ffff1a95001c
           ...
in bounds: false
   length: 12
           ...
Segmentation fault (core dumped)
```

In the trace above, the four Morello instructions execute in A64 mode.

Note: In the example above, the hybrid Morello application is linked to the default system C library.

3.2 Capability faults

If a capability fault occurs during execution, the emulator prints:

- Information about the fault.
- The PC value where the fault occurred.
- Details about faulty capability.

By default, Morello IE terminates the application with an appropriate OS signal. To disassemble the binary use the Morello toolchain `llvm-objdump` tool. To locate the instruction that caused the fault, use the reported PC value. Alternatively, you can use the built-in Morello IE debugger.

You can suppress signals sent to the application with the `-no-signal` option.

The following table shows which signals are emulated for each type of capability faults:

Fault	Signal
Capability tag not set	SIGSEGV
Capability is sealed	SIGSEGV
Incorrect capability permission	SIGSEGV
Access out of capability bounds	SIGSEGV
Anything else	SIGSEGV

You can also provide the `-f-debug` option to pause execution immediately after the fault to gather additional information. For more details refer to Interactive debugger (page 24).

When a capability fault occurs, you can use the `f-cpu` option to show information about the emulated CPU state.

3.3 Instruction and memory trace

The tracer instrumentation client included in the Morello IE allows the generation of runtime traces with information about executed instructions and memory accesses. You can enable this by adding the `--trace` option. For memory only trace, provide the `-mem` option, and for instruction trace only, provide the `-instr` option.

By default, code that is invoked directly or indirectly from `main` function is traced when tracing is switched on. In case of a stripped binary, the entire application code is traced.

Important: To ensure correct execution, code inside a region defined by load-exclusive and store-exclusive instructions, is not instrumented and is not traced.

To change the scope of tracing, use one of the following mutually-exclusive options:

- To force tracing of all application code, use the `-all` option.
- To enable use of tracer marker instructions (see Markers for tracing (page 15)), use the `-roi` option.
- To trace only purecap (C64) code, use the `-c64` option.
- To trace only code executed from the instruction at address `<pc1>` to the instruction at address `<pc2>` (inclusive), use a pair of options `-fr <pc1> -to <pc2>`.
- To trace code called directly or transitively from a given function, use `-main <fun>` option. The default value of this option is `main`.

Note: Provide the `<pc>` values as hexadecimal integer literals with or without `0x` prefix.

The following example shows a trace output generated by Morello IE:

```
#    Seq PC     Mode   Encoding Opcode Operands
M      1 2342ad (C64): 028343ff sub    Cd=31 Cn=31 imm12=208 CapSubImm_LSL0
M      2 2342b1 (C64): 4285fbfd stp    Ct=29 Ct2=30 Rn=31 imm7=11 CapStorePairImmPre
#                      Transfer size   Address          Data
M        2342b1 (C64): MW32            0000ffffcdeb1f70 ffffc0000001 ... fffcdeb1f90
M        2342b1 (C64): CW02            0000ffffcdeb1f70 10
#        ...
M     15 2342e5 (C64): c2c838a5 scbnds Cd=5 Cn=5 imm6=16 CapSetBoundsImmNoS
A     16 2342e9 (C64): 2a1f03e8 orr    %wzr %wzr lsl $0x00 -> %w8
A     17 2342ed (C64): b9000008 str    %w8 -> (%x0)[4byte]
A        2342ed (C64): MW04            0000ffffcdeb1f6c 00000000
```

Every instruction is shown with a type (`M` for Morello instruction and `A` for A64 instruction), a sequential number (`Seq`), an address (`PC`), a mode of execution (`A64` or `C64`), 32-bit encoding value, opcode, and instruction operands.

The emulator prints the instruction trace entry before it executes the instruction. When an error occurs, if you have correctly configured the scope of tracing, the last instruction in the trace is the faulty instruction.

For memory accesses, additional information is provided when memory tracing is enabled:

- Write or read: `MW` or `MR` correspondingly.
- Transfer size in bytes: for example, `MW32` means write of 32 bytes and `MR04` means read of 4 bytes.

- Address used for memory access (64-bit value).
- Data which is loaded or stored.
- For loading and storing capabilities, tags are also shown in binary format. For example, `CW02` with data `10` means that 2 tags have been written, one tag is `1` and the other tag is `0`.

A memory trace entry always follows the corresponding instruction trace entry. When an instruction fails to execute, a memory trace entry is not shown.

3.4 Interactive debugger

The Morello IE includes a simple interactive debugger that provides access to emulated state and runtime context of the executed application.

Entering debug mode

To use the interactive debugger, the Morello emulation client must be in debug mode. There are several ways to do this.

You can insert a breakpoint from the command line. Add the `-break` option with the value of the PC address at which the breakpoint must be inserted. For example:

```
$ morelloie -break 2342d4 -- ./app
```

Note: Provide the `<pc>` value for `-break` option as a hexadecimal integer literal with or without `0x` prefix.

During runtime instrumentation the Morello IE inserts code to pause the execution and to enter debug mode. Debug mode allows you to submit commands to request information about the current execution context. When execution is about to reach an instruction at given address, the application stops and waits for further user commands.

If the `-f-debug` options is used, every capability fault automatically results in entering debug mode. In this case, a faulty instruction has been attempted but has failed. You can obtain information about the current execution context. However, as the emulator leaves debug mode, execution is interrupted with an appropriate OS signal sent to the application.

Finally, breakpoint marker instructions can be inserted in source code using this macro (see Marker for debugger (page 15) for details).

```
/* Put breakpoint */
#define __MIE_DEBUG() __asm__ volatile ("hint #0b1000100")
```

During runtime instrumentation, for every such instruction, Morello IE inserts code to pause the execution and enter debug mode. To enable this, you must add the `-debug` option.

Note: The `-debug` option is implied when either `-f-debug` or `-break <pc>` options are used.

Debugger commands

This section describes commands. In the following commands and examples:

- The `<pc>` placeholder refers to a PC address in form of a hexadecimal literal with or without `0x` prefix.
- The `<addr>` placeholder refers to memory address in form of a hexadecimal literal with or without `0x` prefix.
- The `<reg>` placeholder is a register name, for example `X29` or `CSP`.
- The `<sz>` placeholder is the number of bytes to read from memory (a decimal integer literal).
- The `<type>` placeholder is a type name for loading data from memory, for example `float` or `uint64`.

These are the commands available in debugger.

`m, h, help`
: Print help message.

`q, quit, exit`
: Terminate application and exit.

`c, continue`
: Disable debugging and continue execution ignoring any existing breakpoints. Execution continues until the process exits or a fault occurs.

`r, run`
: Run process until the next breakpoint or until the process exits or a fault occurs.

`s, n, next, step`
: Step to the next instruction (step in).

`finish`
: Run until exit from current function (step out).

`until <pc>`
: Set new breakpoint address to `<pc>` and run until it.

`bt, backtrace`
: Show backtrace (last call shown first, see Backtrace (page 26) for details).

`w, where`
: Show information about current and previous PC addresses.

`info cpu, cpu`
: Print CPU state (emulated and hardware registers).

`p <reg>, print <reg>`
: Print current value of a register: `XSP`, `CSP`, `LR`, `CLR`, `PCC`, `DDC`, `X0` to `X30`, `C0` to `C30`.

`mem <sz> <addr>, mem <sz> <reg>`
: Read `<sz>` bytes from memory address `<addr>` (maximum 1024 bytes can be read). You can also specify the address as the current value of a register (for example, `mem 16 csp` will attempt to load 16 bytes from the top of the current stack).

`mem <type> <addr>, mem <type> <reg>`
: Read data of type `<type>` from memory address `<addr>`. Supported types are: `float`, `double`, `int64`, `uint64`,

`int32`, `uint32`, `int16`, `uint16`, `int8`, `uint8`. The output is formatted according to the specified type.

`cstr <sz> <addr>, cstr <sz> <reg>`
: Read `<sz>` chars from memory address `<addr>` as string. The string is printed up to the first null character.

`tag <addr>, tag <reg>`
: Read memory tag for memory address `<addr>`.

`cap <addr>, cap <reg>`
: Display capability that can be loaded from address `<addr>`.

`br l, br list, breakpoint list`
: List existing breakpoints (their PC addresses and functions containing them).

`br set <pc>, br add <pc>, breakpoint set <pc>, breakpoint add <pc>`
: Create a new breakpoint at `<pc>`.

`br del <pc>, br remove <pc>, br delete <pc>, breakpoint del <pc>`
: Delete existing breakpoint at `<pc>`.

`br c, br clear, breakpoint clear`
: Delete all breakpoints.

Backtrace

Backtrace is a sequence of function calls with information about caller address, callee address, start address of the calling function, mode of execution (A64 or C64), and frame pointer. If available, backtrace provides the name of the function.

For example:

```
# Caller Mode  Frame pointer    Calling function
- 2342c0 (C64) 0000ffffc168df60 2342ac:main
- 234590 (A64) 0000ffffc168e0b0 2343b0:__libc_start_main
- 234188 (A64) 0000ffffc168e0b0 234160:_start
```

This example backtrace shows three function calls: `_start` calls `__libc_start_main` which in turn calls `main`. Start addresses of the functions are `234160`, `2343b0`, and `2342ac`, respectively. In function `_start`, the call to `__libc_start_main` occurs at address `234188`, and the call of `main` from `__libc_start_main` occurs at `234590`. The current PC value is `2342c0` (which is the top-most element of the call stack). Finally, `_start` and `__libc_start_main` are executed in A64 mode, because they are part of non-Morello C library. However, `main` is executed in C64 mode, because it is part of the purecap Morello application.

Use the `where` command to traverse the execution path. This command prints the current PC address, as well as the target and source of the most recent jump instruction. It also displays whether or not this jump was a function call. In addition, the `where` command shows the current link register value. For example:

```
[debug 210364] where
 - {jumped from}   = 2105d4 (A64) [in `libc_start_main_stage2`]
 - {jumped to}     = 21035c (C64) [in `main`]
 - {current pc}    = 210364 (C64) [in `main`]
 - {link register} = 2105d8 (A64) [in `libc_start_main_stage2`]
```

(continues on next page)

(continued from previous page)

```
 - {call}         = yes (whether last jump was a call)
```

This information can be used to trace back to a point of interest when no other means of tracing are available. In the example above you can see that the most recent jump was a function call. You can also see that the `main` function was called from the `libc_start_main_stage2` function, and execution was linear since the start of the `main` function (`21035c`) till the current PC value (`210364`). When current function finishes, we will return to address `2105d8` (current link register value) in `libc_start_main_stage2`. You can also see that execution is done in C64 (purecap) mode since entering the `main` function.

CPU state

The `cpu` or `info cpu` debugger commands show the currently emulated and the actual CPU state. This includes all general purpose capability and aliased AArch64 registers, some system of the registers available at EL0, and finally the process ID and current thread ID information.

```
 # Reg   Emulated value                          Hardware value     Sync
 - C00 = 0x1:ffffc000:5f40df3c:0000ffff:c168df3c (0000ffffc168df3c) [=]
 - C01 = 0x1:97ffc000:40200000:00000000:492e0000 (00000000492e0000) [=]
 - C02 = 0x0:00000000:00000000:00000000:00000000 (0000ffffc168e0c8) [ ]
 - C03 = 0x0:00000000:00000000:00000000:00000000 (00000000002342ad) [ ]
...
 - C30 = 0x1:ffffc000:00010005:00000000:00234594 (0000000000234594) [=]
 - CSP = 0x1:ffffc000:00010005:0000ffff:c168de90 (0000ffffc168de90) [=]
 - PCC = 0x1:ffffc000:00010005:00000000:002342c1
 - CID_EL0         = 0x0:00000000:00000000:00000000:00000000
 - CTPIDR_EL0      = 0x0:00000000:00000000:00000000:00000000
 - CTPIDRRO_EL0    = 0x0:00000000:00000000:00000000:00000000
 - PSTATE.C64      = 0
 - PSTATE.NZCV     = 0000 (0110)
 - CCTLR           = SBL = 0 ADRDPB = 0 DDCBO = 0 PCCBO = 0
 - {process id}    = 19647
 - {thread id}     = 19647 (main thread)
```

Both emulated and real values are displayed for registers `C0` to `C30` and `CSP`. The real 64-bit hardware value is shown in brackets after the emulated 129-bit value. When lower 64 bits of emulated capability register are equal to the real value of the aliased register, the `[=]` symbol is shown on the right. This symbol indicates that the emulated and real values have been synchronised. However, this synchronisation might not always occur because it happens only when necessary for the next instruction to be executed (lazy synchronisation).

If a capability register is used in the next instruction as a source register, its value shown by the debugger is architecturally valid and the lower 64 bits are the same as 64 bits of the value of the corresponding hardware register. This is indicated with `[=]` symbol. However, if a capability register has not yet been used as a destination or a source operand, the emulated CPU state may contain its initial value. In this situation the real hardware register might have a different value set previously by a non-Morello instruction. Capability registers used as destination operands receive an architecturally correct value immediately after execution of the instruction. Lower 64 bits of its value are propagated to the corresponding hardware register immediately.

To see more detailed information about each register, use the `p <reg>` debugger command. For capability registers this gives detailed information about capability, for example:

```
C01 = 0x1:97ffc000:40200000:00000000:492e0000
         tag: true
       value: 0x 00000000492e0000
      offset: 0x0000000000000000
        base: 0x000000000492e0000
       limit: 0x000000000492e0020
       valid: true
   in bounds: true
      length: 32
      offset: 0
 permissions: global ... load
      sealed: false
      flags: 0x 0000000000000000
```

For hardware registers, the 64-bit value is printed.

3.5 Micro-architectural statistics and cache model

The results collected by statistics and cache model modules are printed to `stderr`. By default, simple format is used. In addition it also supports `-format csv` option.

Micro-architectural statistics

The tracer instrumentation client included in the Morello IE can collect runtime information about executed instructions and memory accesses including Morello-specific data such as memory tag operations:

- *Total instructions executed* (including *Morello* instructions and *C64* instructions counter separately).
- *Total writes to memory* (including *Morello* writes and *C64* writes counter separately, and also including writes of each transfer size counted independently for 1, 2, 4, 8, 16, and 32 bytes).
- *Total reads from memory* (including *Morello* reads and *C64* reads counter separately, and also including reads of each transfer size counted independently for 1, 2, 4, 8, 16, and 32 bytes).
- *Tag writes and reads* of tags for capabilities in memory.

You can use the statistics gathering module to analyse non-Morello AArch64 applications as well. Use the `-no-mie` option to disable emulation of Morello code when executing AArch64 applications.

Note: The options controlling the scope of tracing, such as `-roi` or `-c64`, also control the scope of gathering of statistics (see Frontend options (page 9) for more details).

Cache model

The cache model included in Morello IE is part of the tracer instrumentation client. It can be used to collect memory accesses and gather data related to use of CPU cache. The cache model is based on the `drcachesim` tool from DynamoRIO and has limited configurability. It has two levels of cache with independent instruction and data cache at level 1. You can adjust the size of each cache, the size of cache lines, and associativity of the L1 and L2 caches with command line options. For example:

```
morelloie -cache -l1-i-size 1024 -l1-d-size $((32*1024)) -l2-size $((128*1024)) -- ./app
```

This example sets the L1 instruction cache size to 1 KiB, L1 data cache size to 32 KiB and L2 cache size to 128 KiB (see the description of cache model options in Frontend options (page 9) for all the options controlling cache model).

The data returned from the cache model includes information such as number of hits and misses at L1 and L2, miss rates, and the number of executed instructions. When running Morello applications, both AArch64 and Morello instructions and memory access are taken into account.

You can also use the cache model to analyse non-Morello AArch64 applications. Use the `-no-mie` option to disable emulation of Morello code when executing AArch64 applications.

Note: The options controlling the scope of tracing, such as `-roi` or `-c64`, also control the scope of collection of memory references submitted to the cache model (see Frontend options (page 9) for more details).

3.6 Verbose logging

When the option `-verbose` is used, Morello IE produces extra diagnostic messages that are redirected to `stderr`. This includes inexact bounds used for dynamic memory allocation and various runtime warnings.

To show the processed capability relocations (`capdesc` relocation entries), you must add the option `-verb-capdesc`. Each entry will include location, base, offset, size, and permissions as well as the resulting capability. For example:

```
cap desc {location = <loc> base = <base> offset = 0 size = 16 permissions = <perm>} --> <capability>
```

To show the detected and wrapped `C64` thunks which are necessary for calling A64 code from C64 code, you must add the `-verb-libc-thunk` option. For example:

```
wrapping `main` at 21036c
wrapping `__C64ADRPThunk_malloc` at 21120c
```

To see relocated and cleared memory tags (for example, when using the `realloc`, `memmove` or `free` functions), use the `-verbose` option. For example:

```
210490: __C64ADRPThunk_realloc: relocated 1 mem tags from <addr> (<sz> bytes) to <addr>
```

The `-verbose`, `-verb-capdesc` and `-verb-libc-thunk` options can be helpful for diagnostic purposes.

4 Annex

4.1 Building Morello LLVM for AArch64 Linux hosts

This describes how to build a Morello LLVM toolchain natively from source[8]. This relies on using the existing LLVM toolchain of version 9.0.x or later (the “host LLVM”) and can be used on an AArch64 Linux system.

Setup:

```
# where host LLVM is installed
$ export HOST_LLVM_BIN=/path/to/host/llvm/bin
# path to sources of the Morello toolchain
$ export LLVM_PROJECT="$(pwd)/llvm-project"
# target installation path for the Morello toolchain
$ export MORELLO="${HOME}/morello"
```

Get sources:

```
$ git clone https://git.morello-project.org/morello/llvm-project.git
```

Configure:

```
$ mkdir build && cd build
$ cmake \
   -DCMAKE_C_COMPILER=${HOST_LLVM_BIN}/clang \
   -DCMAKE_C_COMPILER_WORKS=YES \
   -DCMAKE_CXX_COMPILER=${HOST_LLVM_BIN}/clang++ \
   -DCMAKE_CXX_COMPILER_WORKS=YES \
   -DCMAKE_AR=${HOST_LLVM_BIN}/llvm-ar \
   -DCMAKE_RANLIB=${HOST_LLVM_BIN}/llvm-ranlib \
   -DCMAKE_NM=${HOST_LLVM_BIN}/llvm-nm \
   -DCMAKE_LINKER=${HOST_LLVM_BIN}/ld.lld \
   -DCMAKE_OBJDUMP=${HOST_LLVM_BIN}/llvm-objdump \
   -DCMAKE_OBJCOPY=${HOST_LLVM_BIN}/llvm-objcopy \
   -DCMAKE_EXE_LINKER_FLAGS="-fuse-ld=lld" \
   -DCMAKE_SHARED_LINKER_FLAGS="-fuse-ld=lld" \
   -DLLVM_TARGETS_TO_BUILD="AArch64" \
   -DCMAKE_BUILD_TYPE=Release \
   -DCMAKE_EXPORT_COMPILE_COMMANDS=ON \
   -DBUILD_SHARED_LIBS=ON \
   -DCMAKE_SKIP_BUILD_RPATH=OFF \
   -DCMAKE_INSTALL_RPATH=\$ORIGIN/../lib \
   -DCMAKE_BUILD_WITH_INSTALL_RPATH=ON \
   -DLLVM_ENABLE_ASSERTIONS=ON \
   -DLLVM_ENABLE_LIBCXX=ON \
   -DLLVM_ENABLE_LLD=ON \
   -DLIBCXX_CXX_ABI=libcxxabi \
   -DLIBCXX_CXX_ABI_INCLUDE_PATHS="${LLVM_PROJECT}/libcxxabi/include" \
   -DLIBCXXABI_USE_LLVM_UNWINDER=ON \
   -DLIBCXX_USE_COMPILER_RT=ON \
   -DLIBCXXABI_USE_COMPILER_RT=ON \
   -DLIBCXX_ENABLE_THREADS=ON \
```

(continues on next page)

[8] https://git.morello-project.org/morello/llvm-project

(continued from previous page)

```
  -DLIBCXXABI_ENABLE_THREADS=ON \
  -DLIBUNWIND_ENABLE_THREADS=ON \
  -DCMAKE_INSTALL_PREFIX=${MORELLO} \
  -DLLVM_ENABLE_EH=ON -DLLVM_ENABLE_RTTI=ON \
  -DLLVM_ENABLE_PROJECTS="clang;lld;lldb;libcxx;libcxxabi;compiler-rt;libunwind" \
  ${LLVM_PROJECT}/llvm
```

Build:

```
$ make -j16
$ make install
```

Compiling crtbegin and crtend objects

Important: When linking to the Morello version of the C library, you must use purecap versions of these objects.

To build purecap Morello applications, you must build your startup code in purecap mode as well. This includes the `crtbegin` and `crtend` objects which are provided by the toolchain.

```
$ ${MORELLO}/bin/clang -march=morello+c64 -mabi=purecap \
    -c ${LLVM_PROJECT}/compiler-rt/lib/crt/crtbegin.c \
    -o ${MORELLO}/lib/clang/11.0.0/lib/linux/clang_rt.crtbegin-morello.o

$ ${MORELLO}/bin/clang -march=morello+c64 -mabi=purecap \
    -c ${LLVM_PROJECT}/compiler-rt/lib/crt/crtend.c \
    -o ${MORELLO}/lib/clang/11.0.0/lib/linux/clang_rt.crtend-morello.o
```