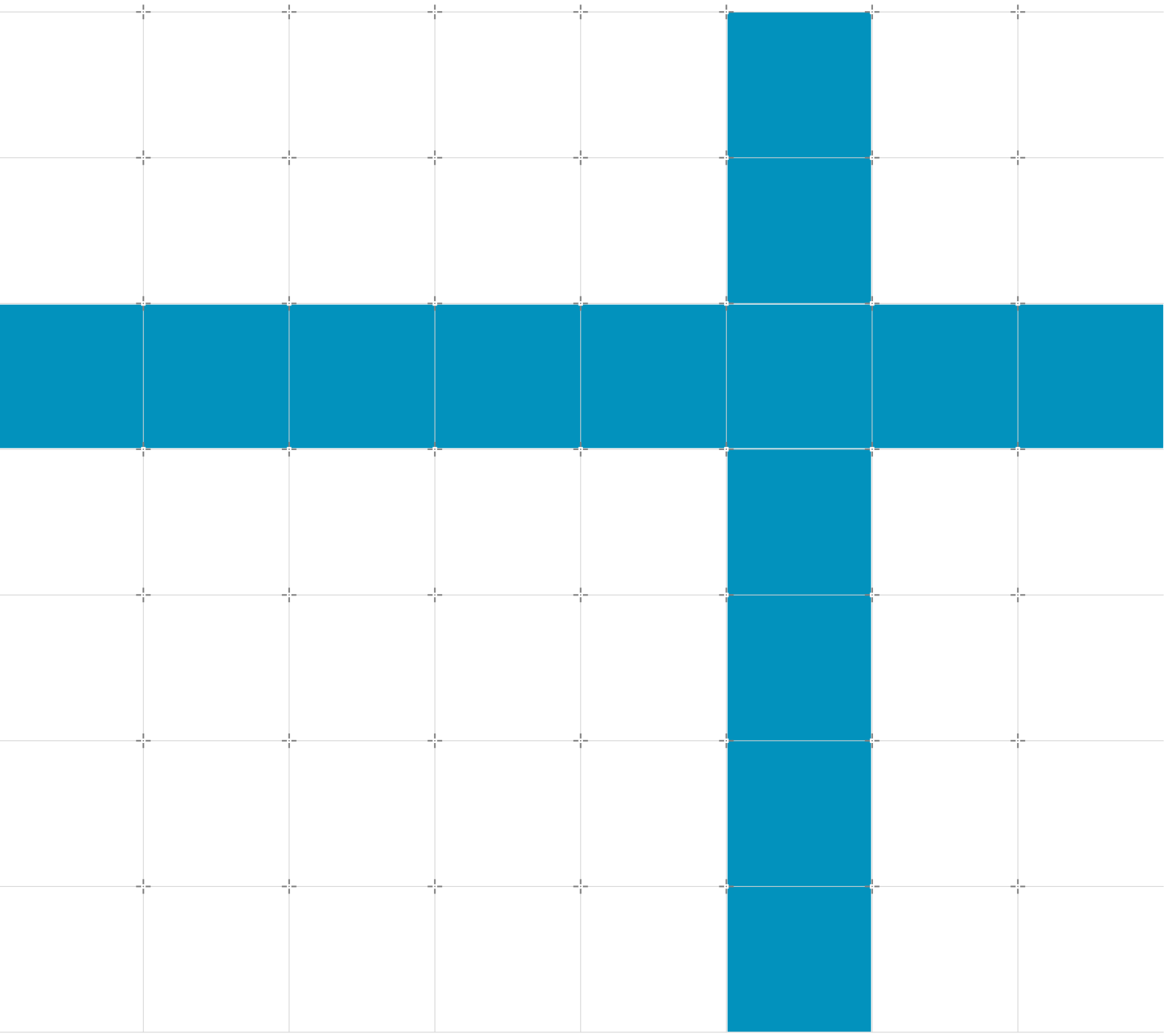
arm

Morello Instruction Emulator User Guide

Non-confidential

Issue

102270 0.6



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.*
0.4	January 2022	Non-Confidential	Updated command line options and debugger commands. Updated Morello LLVM build instructions. *Morello IE release 1.4.*
0.5	July 2022	Non-Confidential	Updated command line options ad debugger commands. Deprecated some command line options. Updated description of the installation procedure. Updated Emulator launcher binary usage. Removed description of the C library emulation layer. *Morello IE release 2.0.*
0.6	October 2022	Non-Confidential	Added new debugger commands. Added description of the new remote mode for debugger. Added new command line options for remote debugger mode. Added description of the new experimental remote debugger clients. *Morello IE release 2.1.*

Proprietary Notice

License

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: 2.1.

1.2 Intended audience

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: Morello IE is an experimental tool. Do not use the Morello IE to run applications in a production environment.

1.3 System requirements

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

- Arm v8.0 64-bit 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.
- Installation script requires `bash`, `tar` and `more` tools.
- Remote Python-based client for the debugger requires either Python 2.7 or Python 3.6 or above.
- Experimental web-based client for the debugger requires up-to-date version of either Mozilla Firefox or a Chromium-based browser.

1.4 Installation

The Morello IE installer is a shell script `morelloie-${VERSION}.tgz.sh` that you should execute in a Bash shell in order to install the tool. The shell script will guide you through EULA acceptance, allow you to choose the installation directory and unpack the contents of the distribution bundle to it. The script supports the following command line options:

- `--prefix=<path>` – this option provides a way to override the path to the installation directory.
- `--i-agree-to-the-contained-eula` – this option supports unattended installation process and allows accepting EULA via a command line option.

To use the emulator launcher, you may add the path to the `bin` directory of the installation root to your `PATH` environment variable. The installation is self-consistent and portable. For example:

```
$ bash morelloie-${VERSION}.tgz.sh [--prefix=/path/to/installation/directory]
  ...
$ export PATH=${PATH}:/path/to/installation/directory/bin
$ morelloie --version
```

Morello architecture is backwards compatible with AArch64, and you can run an AArch64 application with the emulator. For example, 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 binary and remote debugger client.
- `lib` directory with instrumentation clients.
- `license_terms` directory with licence information.
- `README` a short readme file with a brief description recent changes and links to related resources.

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][1]	This document	102270
[Morello ISA][2]	Morello Prototype Architecture Specification	DDI0606
[Morello AAPCS][3]	Morello extensions to PCS for the Arm 64-bit Architecture	102205
[Morello AAELF][4]	Morello extensions to ELF for the Arm 64-bit Architecture	102272
[Morello LLVM][5]	Morello extensions to ELF for the Arm 64-bit Architecture	102216
[CHERI][6]	CHERI C/C++ Programming Guide	
[Linux Toolchain][7]	Guide for Morello LLVM toolchain and Musl libc for Linux.	

[1] https://developer.arm.com/documentation/102270/latest
[2] https://developer.arm.com/documentation/ddi0606/latest
[3] https://github.com/ARM-software/abi-aa/blob/main/aapcs64-morello/aapcs64-morello.rst
[4] https://github.com/ARM-software/abi-aa/blob/main/aaelf64-morello/aaelf64-morello.rst
[5] https://git.morello-project.org/morello/llvm-project-releases/-/blob/morello/release-docs-1.4/UserGuide.pdf
[6] https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-947.pdf
[7] https://git.morello-project.org/morello/musl-libc/-/blob/morello/master/README.rst

2 Overview

2.1 General description

Morello Instruction Emulator (Morello IE) is a dynamic binary translation tool. It is based on the runtime translation of each Morello instruction into a series of AArch64 instructions using DynamoRIO[8] instrumentation framework. 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 system.

Morello IE can execute both hybrid and purecap Morello applications. Staring with release 2.0, you must link an application to a Morello-aware C library. For example, you can use port of the Musl C library to Morello[9]. Depending on your application, the C library may need to include the libshim library[10].

Morello IE implements the Morello ISA of version PROTO_REL_04.

2.2 Components

Morello IE includes the following components:

- Emulator – implements emulation of Morello architecture.
- Debugger – provides debugging capabilities with access to the emulated CPU state and memory capability tags.
- Instruction and memory tracer, cache model and statistics counter.
- Morello IE launcher – this application is used to load all required instrumentation and run the payload.

Emulator. 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 memory allocations that are managed via `mmap` syscalls. The emulator is implemented in the instrumentation client `libmie.so`.

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 maintained by the emulator that is not available by means of `lldb` or `gdb`. See Interactive debugger (page 21) for more information. Since release 2.0 the debugger is implemented as a separate instrumentation client `libdbg.so`.

Tracer. Instruction and memory access tracer is implemented as a separate instrumentation client `libtracer.so`. You can use this client independently of the instruction emulation client. For example, you can use it to analyze non-Morello AArch64 applications to compare the results to the Morello version of the same application. 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.

The tracer instrumentation client also includes a module for collecting various architectural statistics at runtime, including counters specific for Morello, and also data related to CPU cache modelling based on DynamoRIO's `drcachesim` tool. Statistics and cache modelling data can be scoped in the same way as instruction and memory traces (see Command line options (page 8) for more details).

[8] https://dynamorio.org/
[9] https://git.morello-project.org/morello/musl-libc/
[10] https://git.morello-project.org/morello/android/platform/external/libshim

Launcher. Running Morello applications with instrumentation requires an additional launcher binary. The `morelloie` binary loads emulation and tracer clients, and exposes application execution to the dynamic binary translation instrumentation.

You can also use this binary to load additional instrumentation client libraries. For example, this could be useful when you use a custom DynamoRIO instrumentation client along with the Morello emulation client.

2.3 Command line options

This section gives an overview of all the command line options supported by Morello Instruction Emulator. The command line template is:

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

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

Morello IE supports *boolean* (no value required), *integer*, and *string* types of command line options. It also supports program counter (PC) values that you can specify in several ways.

In general, you can invert any boolean option by adding `-no` directly in front of the option name. For example, use `-no-enable-foo` to invert option `-enable-foo`. For options that have a value, separate the option name and the value with a single space.

Some options take a PC address `<pc>` as a value. This can be specified in several formats. It can be a hexadecimal literal (without the `0x` prefix since decimal integers are not supported as PC address values). For example, `-break 200200`. It can also take a form of `<symbol>` or `<symbol>+<offset>`, where `<symbol>` is the name of a symbol in the binary which start address is used as a value for the option and `<offset>` is an optional unsigned decimal offset in bytes added to the start of the symbol address. For example, `-fr main -to main+128`.

Options related to memory sizes accept values as decimal integers and are measured in bytes. For example, `-l1-d-size 1024` means size of 1024 bytes (1 KiB).

Options to control instrumentation

`-v`

Print verbose output from the launcher. The output is directed to `stderr`.
Default: verbose output is suppressed.

`-f`

Use the `fork` syscall instead of `execve` to launch the process.
Default: the `execve` syscall 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 the performance of instrumentation, but might also result in exceeding internal code cache limits in DynamoRIO.

`-Dr,<option>` `[<value>]`

Supply raw option for DynamoRIO. This option can be used multiple times to provide several options that are relayed to

DynamoRIO. The `-fsz <n>` option is a shortcut for the `-Dr,max_bb_instrs <n>` option.

`-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.

Default: Morello emulation client is enabled by default.

`-c` `<lib> [params]`

Load custom DynamoRIO instrumentation client from library `<lib>` with optional parameters. This option should be used after all other launcher options and can be repeated multiple times (to load several clients). The `-c` option starts command line for each instrumentation client and must be followed by the path to the client's library and optionally by the arguments intended for this client. All options between the `-c` option 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.

Common client options

`-verbose`

Show extra diagnostic messages.

Default: verbose output is suppressed.

`-debug`

Enable debug support in all loaded clients. This option is automatically implied when using one of the the debugger client is loaded. This option must be used to enable tracking of instruction markers for the debugger (see Marker for debugger (page 14)).

You can use this option to enter the debug mode on a capability fault.

Default: support for debugging is disabled by default.

Options for Morello emulation client

`-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-strict-cap`

Combine all three `-no-strict-a64-store`, `-no-strict-c64-mem`, and `-no-strict-pcc`. Added in Morello IE 2.0.

Default: all capability checks are 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 19)). This option can be used to override this behaviour. Added in Morello IE 1.2.

Default: signals are emitted for every capability fault.

-DDCBO

-PCCBO

-ADRDPB

-no-SBL

These options control the initial values of the control bits in the CCTRL_EL0 emulated control register: bits `DDCBO`, `PCCBO`, `ADRDPB` are unset (have value 0) by default, bit `SBL` is set (has value 1) by default. Added in Morello IE 1.3.

Options for debugger

-break <pc>

-break <symbol>[+<offset>]

-break <pc-expression-1>,<pc-expression-2>,...

Pauses execution immediately *before* executing the instruction at the given `<pc>` address and enters debug mode. Only a single instance of this option can be used on a command line but this option can accept comma-separated list of PC expressions. Added in Morello IE 1.3.

Default: unset. **Type:** PC (accepts hexadecimal value of symbol names with optional offsets).

-debug-mode local

-debug-mode remote

Select mode for debugger input. When the standard input is used for debugger commands, the default mode is ‘‘local”. Use the `remote` mode to enable HTTP endpoint for debugger commands. See (see Remote debugger (page 24) for more details about the usage of the remote debugger client). Added in Morello IE 2.1.

Default: `local`.

-debug-host <hostname>

-debug-host <IP-address>

Allows you to choose the hostname or IP address for the HTTP endpoint binding for the remote debugger mode. This option only has effect when you use the remote debugger. Added in Morello IE 2.1.

Default: `127.0.0.1`.

-debug-port <port>

Allows you to choose the port number for the HTTP endpoint binding for the remote debugger mode. This option only has effect when the remote debugger mode is used. Added in Morello IE 2.1.

Default: `3450`.

Options to control tracer instrumentation client

`-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 architectural statistics.

Default: collection of architectural statistics is disabled.

`-cache`

Enable cache model and collecting associated data.

Default: cache model is not used.

`-format` `csv`

`-format` `simple`

Format for micro-architectural statistics and cache model results. Added in Morello IE 1.2.

Default: `simple`.

`-tid`

Show thread id in the instruction and memory traces. Added in Morello IE 1.4.

Default: `false`.

`-no-stat-merge`

Do not merge statistics from all threads. By default, statistics for all process threads are merged when displaying the results. Use this option to show statistics for each thread separately. Added in Morello IE 2.0.

Default: `true`.

`-stat-breakdown`

Show statistics breakdown by mode of execution (C64 and A64) and ISA (Morello or AArch64). Added in Morello IE 2.0.

Default: `false`.

Options to control scope of tracing

`-trace-symbol` `<symbol>`

A symbol (function) name for the default tracing scope. Tracing will start once the execution enters this function and will stop at the return from this function.

Default: `main`.

Note: The following sets of options are mutually exclusive.

`-all`

Trace all instructions that execute, this includes instructions that execute outside of the function selected by the `-trace-symbol` option.

`-roi`

Trace only instructions from the region(s) of interest which are defined by the tracer marker instructions (see Markers for tracing (page 14)).

`-fr` `<pc1>`

`-to` `<pc2>`

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

Default: unset. **Type:** PC (accepts hexadecimal value of symbol names with optional offsets).

Options for cache model

`-l1-d-size`

L1 data cache size (in bytes), size must be power of two and multiple of the cache line size.

Default: 65536 bytes. **Type:** integer (accepts integer decimal values).

`-l1-i-size`

L1 instruction cache size (in bytes), size must be power of two and multiple of the cache line size.

Default: 65536 bytes. **Type:** integer (accepts integer decimal values).

`-l2-size`

L2 cache size (in bytes), size must be power of two and multiple of the cache line size.

Default: 1048576 bytes. **Type:** integer (accepts integer decimal values).

`-l3-size`

L3 cache size (in bytes), size must be power of two and multiple of the cache line size. Added in Morello IE 1.3.

Default: 8388608 bytes. **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 data cache associativity (number of ways), must be power of two.

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

-l1-i-ways

L1 instruction cache associativity (number of ways), must be power of two.

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

-l2-ways

L2 cache associativity (number of ways), must be power of two.

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

-l3-ways

L3 cache associativity (number of ways), must be power of two. Added in Morello IE 1.3.

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

-cache-prefetcher `nextline`

-cache-prefetcher `none`

Hardware data prefetcher policy: `nextline` or `none`. Added in Morello IE 1.2.

Default: `none`.

-cache-rep-policy `LRU`

-cache-rep-policy `LFU`

-cache-rep-policy `FIFO`

Cache replacement policy (`LRU` – least recently used, `LFU` least frequently used, `FIFO` – first in first out). Added in Morello IE 1.2.

Default: `LRU`.

-cache-show-params

Show cache model parameters in the results. Added in Morello IE 2.0.

Default: cache model parameters are not included in the results by default.

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.

2.4 Inline control instructions

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:

```
morelloie -roi -- ./app
```

You can define macros for the tracer instructions as shown below:

```
/* 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 the `-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

Morello IE can run both purecap and hybrid Morello userspace applications on non-Morello AArch64 Linux systems. However, pay attention to the following exceptions and limitations.

Tracing stripped binaries

By default, in stripped applications the entire execution flow is traced. The `-trace-symbol` option will not have any effect and tracing will fallback to the configuration initialised by the `-all` option.

Use of a non-Morello C library

Starting with release 2.0 Morello IE no longer supports Morello applications linked to non-Morello C library. You should use one of the available Morello C libraries, for example Morello Musl libc[11].

Multi-threaded applications

Support for multi-threaded applications based on `libpthread` is experimental. Most of the use cases will work (subject to correct implementation in your C library), however cases like thread cancellation might not work consistently.

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

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[12] 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 [Linux Toolchain].

The current version of the emulator 2.1 can execute:

- statically linked purecap Morello applications,
- dynamically linked purecap Morello applications (experimental support),
- statically and dynamically linked hybrid Morello applications,
- statically and dynamically linked AArch64 applications.

Note: See the [Morello AAPCS] for more details about the different types of Morello applications (purecap and hybrid) and their execution modes (A64 and C64).

Running an application under 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 command `clang` 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;
}
```

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

```
$ clang --target=aarch64-linux-musl_purecap -march=morello+c64 \
        --sysroot ${MUSL}/include hello.c -o hello --static
```

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

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 emit only Arm v8.0 code. This should not affect Morello functionality, however wherever possible Arm recommends that you use `-march=morello+c64`.

To run this example, use:

```
$ morelloie -- ./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.

Finally, use `-target=aarch64-linux-musl_purecap` and `-march=morello+c64` to instruct the compiler that we are doing cross-compilation to a purecap Morello target.

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 using Morello IE:

```
$ morelloie -- ./stack
21260c: simulated capability fault in thread 83534:
Out of bounds access to 4 bytes: [0000 ... 9d98)
Faulty capability: 0x1:dc104000:5d989d8c:0000ffff:fa0b9d98
          tag: true
        value: 0x00000fffffa0b9d98
         base: 0x00000fffffa0b9d8c
        limit: 0x00000fffffa0b9d98
```

(continues on next page)

(continued from previous page)

```
 ...
Segmentation fault (core dumped)
```

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 in Morello IE:

```
$ morelloie -- ./heap
212684: simulated capability fault in thread 83793:
Out of bounds access to 4 bytes: [0000 ... 00cc)
Faulty capability: 0x1:dc1f4000:40cc00c0:0000fffd:822100cc
          tag: true
        value: 0x00000fffd822100cc
         base: 0x00000fffd822100c0
        limit: 0x00000fffd822100cc
              ...
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. This enables instruction tracing for the

region of interest. For example:

```
$ morelloie -trace -roi -- ./hybrid
M      1 21080c (A64) c24007e0 ldr     c0, [csp, #16]
M        21080c (A64) MR16             0000fffffffe760 --> 0000000000250010 ffffc000401c0010
M        21080c (A64) CR01             0000fffffffe760 --> 1
M      2 210810 (A64) e280c408 ldur    w8, [x0, #12]
210810: simulated capability fault in thread 98325:
Out of bounds access to 4 bytes: [0000 ... 001c)
Faulty capability: 0x1:ffffc000:401c0010:00000000:0025001c
       tag: true
     value: 0x0000000000025001c
      base: 0x00000000000250010
     limit: 0x0000000000025001c
           ...
Segmentation fault (core dumped)
```

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 and thread ID.
- Details about the faulty capability.

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

Note: All emulated capability faults result in the same signal `SIGSEGV`.

3.3 Instruction and memory trace

The tracer instrumentation client included in Morello IE allows you to generate runtime traces with information about executed instructions and memory accesses. To enable tracing, add 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 the function specified by the `-trace-symbol` option (`main` by default) is traced when tracing is switched on. In the 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 14)), use the `-roi` 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 `-trace-symbol <fun>` option.

Every instruction is displayed with the following information:

- a type (`M` for Morello instruction and `A` for A64 instruction),
- a sequential number,
- thread ID (when `-tid` option is used),
- an address (`PC`),
- a mode of execution (`A64` or `C64`),
- 32-bit encoding value,
- opcode,
- 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:

- `MW` for memory write or `MR` for memory read.
- 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.

When running multi-threaded applications, it might be useful to add thread id to the instruction or memory trace to help filter traces originating from different threads. To do this, use `-tid` option. For example,

```
morelloie -tid -trace -- ./app
```

3.4 Interactive debugger

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 <pc>` option with the value of the PC address at which the breakpoint must be inserted. For example:

```
$ morelloie -break 2342d4 -- ./app
$ morelloie -break main+16 -- ./app
```

See Command line options (page 8) for more information about accepted values.

During runtime instrumentation 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 present, other execution threads also suspend.

You can also use breakpoint marker instructions in your source code using this macro (see Marker for debugger (page 14) 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 to enter debug mode. To enable this, you must use the `-debug` option.

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

Debugger commands

This section describes debugger 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. This can also be a function name with an optional unsigned integer offset, for example `main+64`.
- The `<addr>` placeholder refers to a memory address in form of a hexadecimal literal with or without `0x` prefix. You can specify this as the value of the a register with optional signed integer offset, for example `csp-32` or `x0+8`.
- 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.

r, **run**

Run the 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). The application will run until the execution reaches the current link register address.

until <pc>

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

bt, **backtrace**

Show backtrace (last call shown first).

w, **where**

Show information about current and previous PC addresses.

info cpu, **cpu**

Print CPU state (emulated and hardware registers).

info modules

Print information about loaded modules (executable and shared objects if any).

info fun [<pattern>]

Print information about functions and their addresses. This can be filtered using optional wildcard pattern.

info threads

Print information about application threads.

p <reg>, **print** <reg>

Print current value of a register: `XSP`, `CSP`, `LR`, `CLR`, `PCC`, `DDC`, `X0` to `X30`, `C0` to `C30`, etc.

mem <sz> <addr>

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). You can use an optional signed offset with the register value, for example `mem 16 csp+32`.

mem <type> <addr>

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.

frame [n]

Show current (or, if specified, n-th) stack frame (n is `0`-based) as a sequence of capabilities displaying the capability metadata for all valid capabilities in this memory region. If you omit number `n` of the stack frame, the current (lowest) stack frame displays.

view <addr> <addr>

Show the memory region between two specified addresses as a sequence of capabilities displaying capability metadata for all valid capabilities in this memory region.

cstr <sz> <addr>

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

tag <addr>

Read memory tag for memory address `<addr>`.

cap <addr>

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.

set

Show current settings for debugger.

set th on|off

Toggle showing thread ID on / off.

set mod on|off

Toggle showing module name on / off.

set sync on|off

Toggle synchronisation of displayed register values between emulated registers and real hardware registers on / off.

set reg native|emulated

Toggle showing native or emulated registers on / off.

d, **disassemble**

d, **disassemble** <pc>

d, **disassemble** <pc> <length>

Disassemble at the current or given PC address showing at most `<length>` instructions above and below the chosen PC address.

Morello IE maintains emulated capability registers and, when necessary, synchronizes them with real hardware registers. Updates of the destination capability registers propagate to real hardware registers immediately after execution of an instruction. However updates of the real registers that occur during non-Morello instructions are not delivered to the corresponding emulated registers immediately. In such a case the aliased emulated register will only synchronize when it is about to be used as a source register by one of the following instructions.

When checking the CPU state in the debugger, this synchronization may have not yet occurred for all emulated registers. By default, the debugger synchronizes displayed values of the emulated registers. When this is enabled (`set sync on`), the debugger shows registers as they will appear for the next instruction when it begins execution. When this mode is disabled, the value of some emulated registers observed in the debugger may not be correct. However this will not affect the correctness of execution.

3.5 Remote debugger

Morello IE can be configured to run in the remote debugging mode in which the input and output from the application and from the debugger are decoupled. This allows for more convenient debugging when the application requires user interaction. In this mode, the Morello IE starts HTTP endpoint using configurable hostname and port number. This endpoint provides API that can be used in interact with the debugger.

Remote debugger HTTP API

When using the remote debugger mode, the Morello IE provides the following HTTP endpoints:

```
GET /morelloie/command?cmd=${command}
```

Submit a debugger command (must be URL-encoded). The reply includes a numerical response ID that can be used to obtain the result for this command from the debugger. Synchronous, connection is blocked until the debugger has accepted the command.

Description	Sends a command to the debugger and returns a message ID.	
Method	GET	
Encoding	ASCII / UTF-8	
Content Type	text/plain	
Responses	Code	Reason
	200	Command sent to Debugger and response ID returned
	205	The command was recognised as an exit command and the debugger will exit
	400	Missing of malformed cmd parameter
Body	Text - Message ID stored in debugger	

```
GET /morelloie/response?id=${id}
```

Obtains a debugger response for the given request ID. This is synchronous and blocks until the debugger generates the message.

Description	Retrieves a message from the debugger if one exists.	
Method	GET	
Encoding	ASCII / UTF-8	
Content Type	text/plain	
Responses	Code	Reason
	200	Message returned successfully
	400	Missing of malformed id parameter
Body	Text - The message text for the supplied request ID.	

```
GET /morelloie
```

Returns a web page with a simple client for the remote debugger (see Remote debugger client (web-based) (page 26) for more information).

Remote debugger client (Python-based)

The Morello IE provides a Python-based client for interacting with the remote debugger HTTP API. The client script `debugger-client` is located in the `bin` directory. It is a simple wrapper around the HTTP API. It supports the following command line options:

-h, **--help**

Show this help message and exit.

--host <hostname>

Remote debugger hostname or IP address (default: `127.0.0.1`).

--port <port>

Remote debugger port number (default: `3450`).

--timeout <seconds>

Timeout in seconds (default: `10`).

Once connected, this client offers the debugger prompt that accepts all the usual debugger commands (see Debugger commands (page 21) for details).

To exit the client, use either `Ctrl+C` or `Ctrl+D` shortcuts. To stop the debugger and exit, use the usual `exit` debugger command.

You can also use this client programmatically. For example:

```
client = DbgClient('127.0.0.1', 3450)
client.wait_for_server()
client.send('info cpu')
```

Remote debugger client (web-based)

The web-based client for the remote debugger is an experimental tool. You can use this tool to debug Morello applications that are executed by the Morello IE. It provides command prompt for the usual debugger commands as well as automatically updated views into registers, callstack, threads and disassembly for the current PC address.

If you start Morello IE in debug mode and use the remote debugger mode, the web-based client's access URL will display in the output.

3.6 Statistics and cache model

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

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. These metrics are collected:

Metric	Description
Instr count total	Total number of executed instructions
Instr count SVC	Number of executed `SVC` instructions (number of syscalls)
Instr count LDR	Number of executed `LDR` instructions
Instr count STR	Number of executed `STR` instructions
Instr count LDP	Number of executed `LDP` instructions (loads of pairs).
Instr count STP	Number of executed `STP` instructions (stores of pairs).
CTI direct	Number of executed direct branch control transfer instructions (target is part of the encoding)
CTI indirect total	Number of executed indirect branch control transfer instructions (target is a register)
CTI indirect capability	Number of executed indirect branch control transfer instructions with capability targets
CTI conditional	Number of executed conditional control transfer instructions
CTI with link	Number of executed conditional control transfer instructions with link (number of function calls)
Loads N bytes	Number of loads of `N` bytes (`N` can be 1, 2, 4, 8, 16 and 32)
Capability loads	Number of load capability instructions
Capability pair loads	Number of load capability pair instructions
Stores N bytes	Number of stores of `N` bytes (`N` can be 1, 2, 4, 8, 16 and 32)
Capability stores	Number of store capability instructions
Capability pair stores	Number of store capability pair instructions
Loaded bytes	Total amount of loaded data in bytes
Loads count total	Total number of load operations (memory reads)
Stored bytes	Total amount of stored data in bytes
Stores count total	Total number of store operations (memory writes)

In addition, you can use the statistics gathering module to analyse non-Morello AArch64 applications. When you execute AArch64 applications, use the `-no-mie` option to disable emulation of Morello code.

Note: The options controlling the scope of tracing, such as `-roi` or `-c64`, also control the scope of gathering of statistics (see Command line options (page 8) for more details).

You can collect statistics separately for each thread. To do this, use the `-no-stat-merge` option. If necessary, all counters can be broken down by mode of execution (C64 and A64) and ISA (Morello or AArch64). To do this, use the `-stat-breakdown` option.

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. It has three 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 all 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 Command line options (page 8) for all the options controlling cache model).

The data returned from the cache model includes information such as number of hits and misses, hit and 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 Command line options (page 8) for more details).

You can configure the following cache model parameters:

Parameter	Default	Description
CPU cores	`1`	Number of CPU cores
Cache line size	`64`	Size of cache line in bytes
HW prefetcher	`nextline`	Hardware prefetcher (available options: `none` and `nextline`)
Cache replace policy	`LRU`	Cache replace policy: `LRU`, `LFU` or `FIFO`
L1D size	`65536`	Size of L1 data cache in bytes
L1D associativity	`4`	Number of ways in L1 data cache
LID size	`65536`	Size of L1 instruction cache in bytes
LID associativity	`4`	Number of ways in L1 instruction cache
L2 size	`1048576`	Size of L2 unified cache in bytes
L2 associativity	`8`	Number of ways in L2 unified cache
L3 size	`8388608`	Size of L3 unified cache in bytes
L3 associativity	`8`	Number of ways in L3 unified cache

The cache model collects the following metrics:

Metric	Description
L1D hits	Number of hits in L1 data cache
L1D misses	Number of misses in L1 data cache
L1D compulsory misses	Number of compulsory misses in L1 data cache
L1D child hits	Number of child hits in L1 data cache
L1D prefetch hits	Number of hits due to use of HW prefetcher in L1 data cache
L1D prefetch misses	Number of misses due to use of HW prefetcher in L1 data cache
L1D hits	Number of hits in L1 instruction cache
L1D misses	Number of misses in L1 instruction cache
L1D compulsory misses	Number of compulsory misses in L1 instruction cache
L1D child hits	Number of child hits in L1 instruction cache
L1D prefetch hits	Number of hits due to use of HW prefetcher in L1 instruction cache
L1D prefetch misses	Number of misses due to use of HW prefetcher in L1 instruction cache
L2 hits	Number of hits in L2 unified cache
L2 misses	Number of misses in L2 unified cache
L2 compulsory misses	Number of compulsory misses in L2 unified cache
L2 child hits	Number of child hits in L2 unified cache
L2 prefetch hits	Number of hits due to use of HW prefetcher in L2 unified cache
L2 prefetch misses	Number of misses due to use of HW prefetcher in L2 unified cache
L3 hits	Number of hits in L3 unified cache
L3 misses	Number of misses in L3 unified cache
L3 compulsory misses	Number of compulsory misses in L3 unified cache
L3 child hits	Number of child hits in L3 unified cache
L3 prefetch hits	Number of hits due to use of HW prefetcher in L3 unified cache
L3 prefetch misses	Number of misses due to use of HW prefetcher in L3 unified cache
L1D miss rate	Percentage of misses per total number of memory operations for L1 data cache
L1I miss rate	Percentage of misses per total number of memory operations for L1 instruction cache
L2 miss rate	Percentage of misses per total number of memory operations for L2 unified cache
L3 miss rate	Percentage of misses per total number of memory operations for L3 unified cache
Total instructions	Total instructions calculated as number L1I hits plus L1I misses