MINERVA SYSTEMS ARCHITECT USER GUIDE

For AMD Kria™ KV260 Starter Kit

Overview

Minerva Systems Architect helps identify and solve memory-related interference problems during the integration phases of complex mixed-criticality workloads. Architect provides a framework to profile, visualize, and analyze memory accesses performed by applications.

Click here to download the demo.

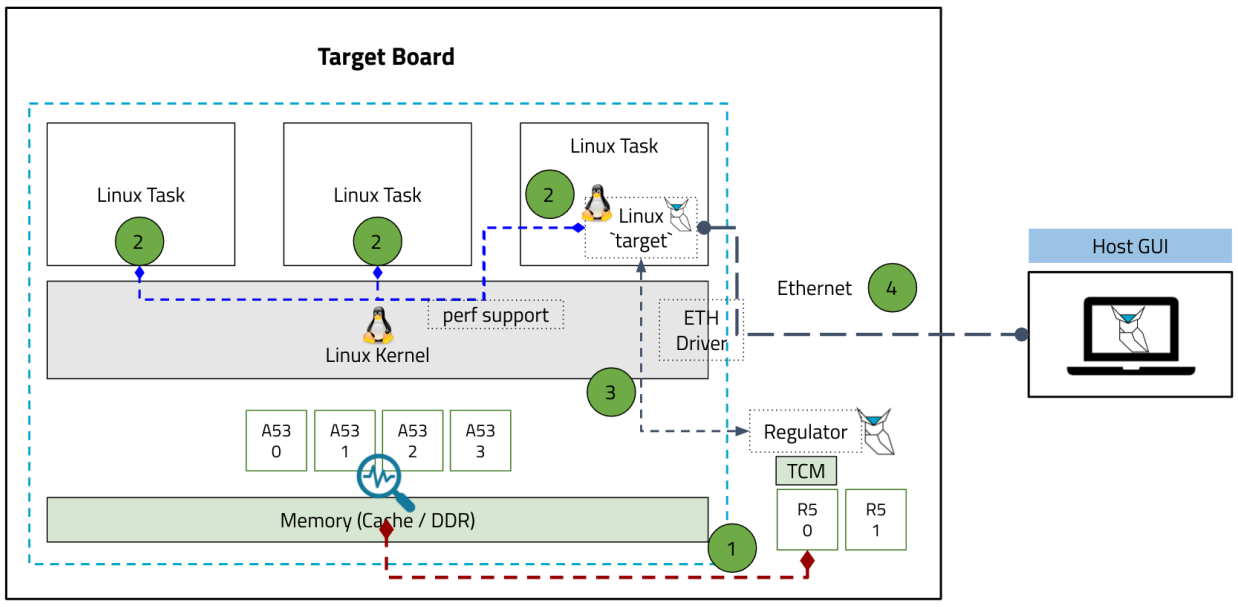
TABLE OF CONTENTS

1 - Introduction

Minerva Systems (MinervaSys) Architect helps identify and solve memory-related interference problems during the integration phases of complex mixed-criticality workloads. The Architect tool provides a framework designed to profile, visualize, and analyze memory accesses performed by user applications. Architect leverages performance counters to enact minimal-overhead and high-resolution profiling. Additionally, it provides features to visualize process-level context switches together with memory accesses, thus enabling users to correlate the running processes with the corresponding memory accesses. Architect can also simulate different types of interference—e.g., memory traffic, cache thrashing, or TLB invalidation—allowing users to understand, at runtime, the sensitivity of applications to different sources of interference. Furthermore, Architect can be used to throttle the memory bandwidth available to each core. This can help with, for example, isolating real-time cores from data-intensive cores if the former are experiencing memory contention.

MinervaSys Architect is composed of two main entities: the 'target' and the 'host.' The host is the GUI, from which you can view the data and communicate with the target. The target is the board on which the backend application is running.

Overview of the System



Architect is composed of software components running on the “target board” (e.g., Kria KV260 Starter Kit) and by the graphical user interface (GUI) running on the developer/integrator Host PC.

On the target board side, the workflow of actions performed by Architect is as follows (identified by the green circles in the picture):

1) PMU-based high resolution monitoring of core-related memory transactions (R5 memory bandwidth regulator and monitor)
2) Linux-perf based monitoring of Linux task/threads (context-switch information, task information, etc.)
3) The `target` program collects and consolidates information from the regulator and perf-tracing
4) The `target` interacts with and delivers information to GUI
    a) Uses Linux Ethernet network driver and TCP/IP stack

1.1 - About this documentation

This document describes the installation and usage of MinervaSys Architect. The document contains a step-by-step guide to install Architect on a target board, how to run the GUI on the host development system, and how to use the Architect to trace, monitor, and regulate applications and memory.

- Chapter 2: How to install the release, both on the target (embedded board) and the host (laptop/ development workstation).
- Chapter 3: How to use Architect.
- Chapter 4: Demonstration of what you can do with Architect and how to do it.
- Chapter 5: How to get in touch with us.

1.2 - Notation conventions

Type	Description
Italics	File and folder names
`Monospaced`	Commands output
`Monospaced bold`	Commands and code snippet

2 - Installation

The release includes both Docker containers and files for manual installation. Specifically, you will find two separate folders for the target and the GUI.
Inside the target folder, you will find an .img file containing the image of a modified Ubuntu/PetaLinux distribution, tailored to enable the R5 core and already containing all binaries, scripts, and guides for a quick target setup.
As an alternative to the .img file installation, all necessary binaries to run Architect are provided. Additionally, the *system.dtb* file is supplied, containing the required modifications to enable the R5 core.
As for the GUI folder, it includes both the Docker container to run the application and the executable. Both solutions have no additional dependencies, so you can choose your preferred option based on personal preference.

Note: in this guide, the Docker container solution is preferred and treated as the primary option.

You can verify the integrity of the downloaded files against the checksums provided in the SHA512SUMS file.

Prerequisites

The host application currently supports only Debian-based distributions and, in particular, has been tested on Ubuntu 20.04 and Ubuntu 22.04.
Since the Docker container solution is the preferred choice, we recommend running Architect using the Docker container. Therefore, the only requirement for running the GUI of Architect is to have Docker installed.

To install Docker, please follow this guide: https://docs.docker.com/engine/install/ubuntu/
After Docker is installed, it is necessary to create an ad-hoc group for it. Please launch the following shell commands:

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

Regarding the target, MinervaSys Architect has been tested on Ubuntu Desktop 22.04 LTS for the Kria KV260 Starter Kit. In order to run the target application, some modifications to the device tree blob (dtb) are required.

Content of the release

The release contains several files for both the target board and the host laptop:

```
├── MINERVASYS ARCHITECT USER GUIDE.pdf
├── gui/
│   ├── assets/
│   └── Architect
├── gui-docker/
│   ├── minerva-architect.tar.xz
│   └── run_gui.sh
├── README.md
└── targets/
    ├── kria/
    │   ├── system-kria.dtb
    │   ├── r5.bin
    │   └── r5.elf
    ├── membw_ctrl
    ├── membw_loader
    └── target
```

2.1 - Target

With “target board” we are referring to the embedded board which will be analyzed by Architect.
The *targets* folder contains everything you need to run the target side of Architect on your preferred supported board.

2.1.1 - Prepare the board

Kria™ SOM - Prepare the board

For the Kria™ KV260 Starter Kit, you can directly use a pre-generated image (*image-kria-app-store.img*), which can be downloaded here. Note: verify the integrity of the download with the command

```
sha512sum -c image-kria-app-store.img.sha512
```

The image can be flashed on the SD card by following this guide.

In order to interact with the Kria™ SOM we are going to use ssh communication. Make sure you connect the Kria™ SOM to the network and you have access to it.

Note: using shell via serial communication is currently not an option, since the R5 core is going to reset the serial during boot and Linux won’t be able to use it again. Serial communication is going to be needed for the first setup though, either to check the IP of the board or to set it. If the Ethernet cable is directly connected to your laptop and the Kria™ SOM does not have an IP, it may be set using a command like:

```
sudo ifconfig eth0 192.168.1.33
```

Note that you also need to change the IP address on the host development system’s side, and that the two IP addresses must belong to the same subnet. If Linux keeps resetting the static IP address, you can disable the network manager service by launching this command:

```
sudo systemctl stop NetworkManager
```

The login user of the provided image will be **ubuntu**, and the password will be **minervasys**.
If you are using the provided image, you can skip to 2.1.2.

If you can’t use the provided image, you can change the *system.dtb* file by following these steps:

1) Follow the official guide Setting up the SD Card Image to prepare the SD card.

2) Rename the provided *system.dtb* file as *user-override.dtb* and move it to the *system-boot* partition of the SD card.
3) Insert the SD card and boot the board.
4) Append *cpuidle.off=1* to LINUX_KERNEL_CMDLINE_DEFAULTS inside */etc/default/flash-kernel* .
5) Reconfigure the flash-kernel package with *sudo flash-kernel* .
6) Reboot the board.

2.1.2 - Load the regulator

The files that you will need from the *target-files* folder are:

- *r5.bin*: A binary needed to enable the regulator on the R5 core
- *r5.elf*: Elf file needed by the Linux firmware loader
- *membw_loader*: A Linux binary that 'loads' the r5.bin file

In order to load the regulator and run the target application, you need to copy the provided files *r5.bin*, *r5.elf*, *membw_loader* to the home folder of the board:

```
sudo -s
cp r5.elf /lib/firmware/
echo -n /lib/firmware > /sys/module/firmware_class/parameters/path
echo r5.elf > /sys/class/remoteproc/remoteproc0/firmware
echo disabled > /sys/class/remoteproc/remoteproc0/recovery
echo start > /sys/class/remoteproc/remoteproc0/state
./membw_loader r5.bin
exit
```

Note: if you don’t see the *remoteproc0* folder, you are probably using the wrong DTB and you will not be able to run all the available features. If this is the case, please also ignore the other steps.

At the end of these steps you should obtain an output like:

```
R5_0: loading r5.bin (10712 bytes)
R5_0: starting core
```

2.2 - GUI

In order to run the GUI, you have two options: the Docker container or the executable. The preferred choice should be the container, in order to avoid any possible dependency issues. Below are the instructions to launch the application, either from the Docker container or by directly using the provided executable. To run the Docker container, you can simply run the *run_gui.sh* script that you will find inside the *gui-docker* folder.

If you don't want to use the Docker container, you need to go inside the *gui* folder and run:

```
LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libstdc++.so.6 ./Architect
```

3 - Usage - SmartCam Demo

An effective demo to showcase what can be achieved with the Minerva Architect utilizes the SmartCam Demo application provided by AMD. SmartCam is an FPGA-accelerated application that enables facial recognition of video input from a webcam or a video source. By using Architect, it is possible to observe how interference can cause a significant slowdown in video processing and facial recognition, potentially leading to disruptions in an industrial application of this software.
If you boot the board using the provided .img you should have the SmartCam application already installed. Otherwise, please follow the instructions on the official GitHub repo to install it manually.

Note: for this demo we used a monitor with resolution 1920x1080p, attached to HDMI.

3.1 Target application

```
usage: target [<options>] [-p <ip-port>] [-s <ip-address>]
[<platform-id>]
```

To run the target on our selected board, we need to launch the target application specifying which platform we are using. For example:

`sudo ./target kria_kv260`

Note that this command will not produce any output, but Architect is running. If you want to see some output, just put `-vvv`.

You can get the list of the supported platforms by running:

`sudo ./target --list-platforms`

If you want to know which version of the target application you are running, just type:

`sudo ./target --version`

You can also bind the target to a specific port or IP address. Note that the default port is 5450.

3.2 SmartCam application

If you have already started the target application, stop it and start it again after the SmartCam application.
In order to run the SmartCam application you can follow the following instructions.

Note: in order to avoid issues with the SmartCam application, please be sure that the HDMI monitor is already attached during the boot phase. If not, please attach it, reboot the board and repeat the previous steps.

```
sudo xmutil desktop_disable
```

Note: this will turn off your desktop, so make sure to be connected also with minicom or ssh.

```
sudo xmutil unloadapp
sudo xmutil loadapp kv260-smartcam

docker run \
      --env="DISPLAY" \
      -h "xlnx-docker" \
      --env="XDG_SESSION_TYPE" \
      --net=host \
      --privileged \
      --volume="$HOME/.Xauthority:/root/.Xauthority:rw" \
      -v /tmp:/tmp \
      -v /dev:/dev \
      -v /sys:/sys \
      -v /lib/firmware/xilinx:/lib/firmware/xilinx \
      -v /run:/run \
      -it xilinx/smartcam:2022.1 bash
```

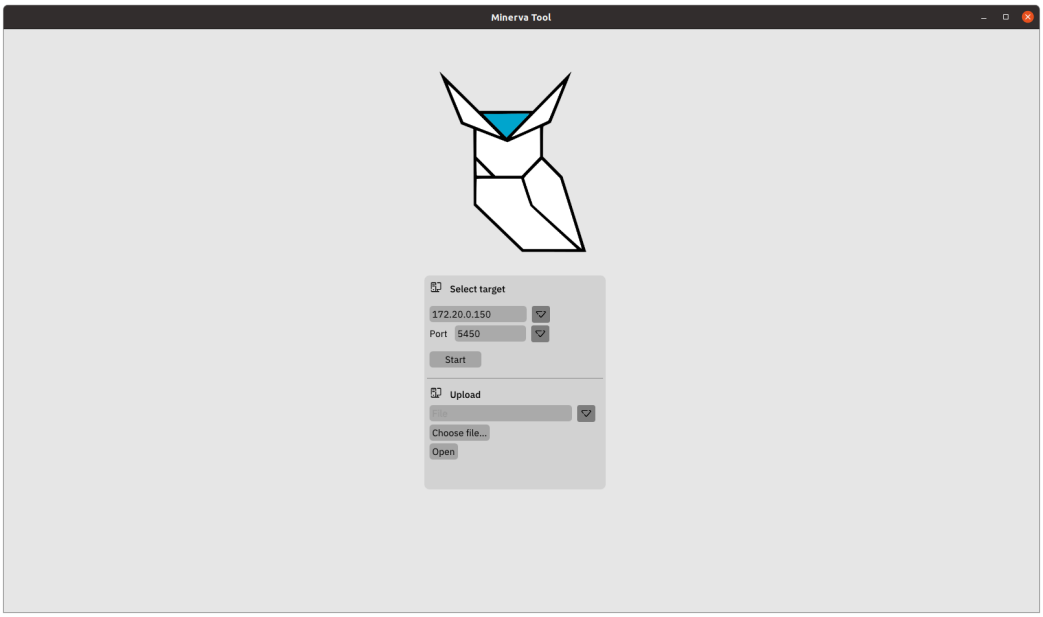
This will run the Docker container and open it. The following commands have to be sent from the Docker's container terminal.

```
echo "firmware:
/usr/lib/firmware/xilinx/kv260-smartcam/kv260-smartcam.xclbin" >
/etc/vart.conf

smartcam --usb 1 -W 1920 -H 1080 -r 30
```

Note: this assumes you have a webcam connected to USB port number 1. Otherwise, please set the correct USB port number or follow the instructions you get using **`smartcam -h`**.

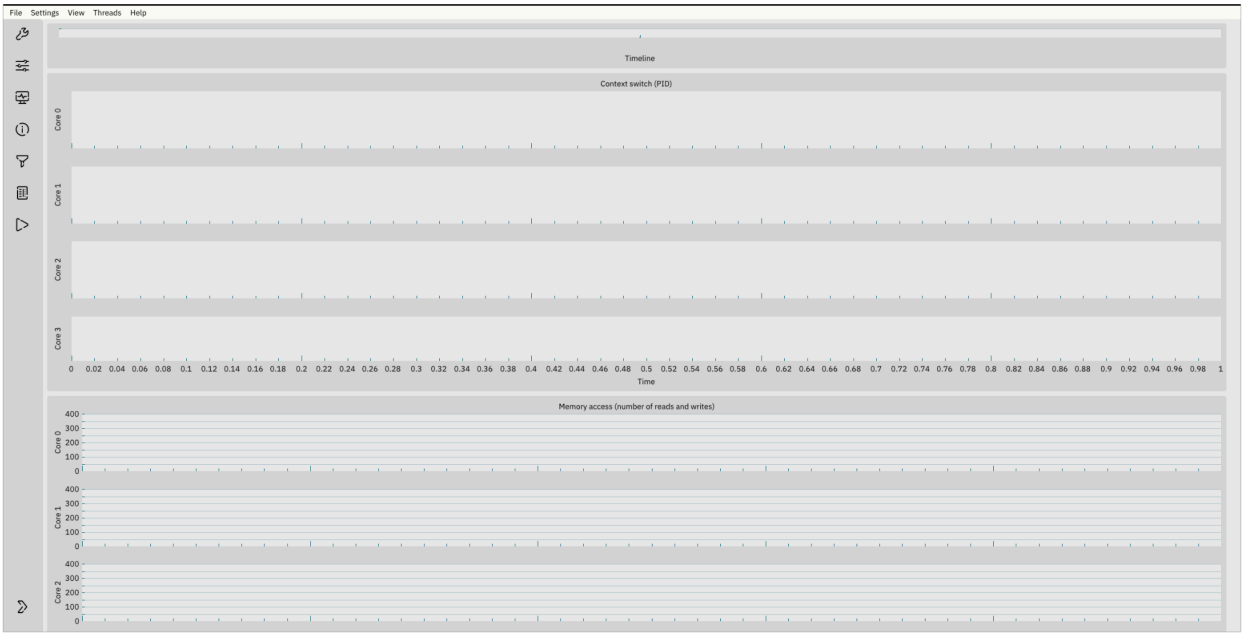
3.3 GUI



In the initial window, you can select the target using its IP address or load an old session from a previously saved file (note: this feature may not be available in your specific version).

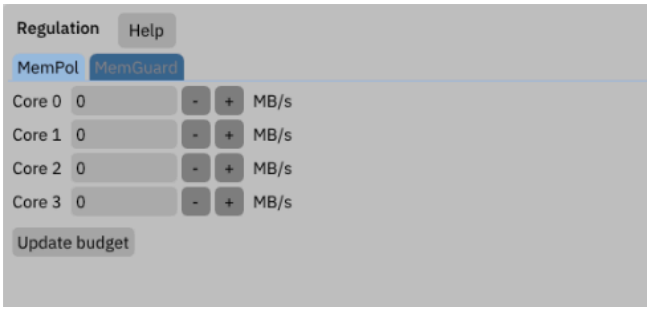
Note: some features described in the following sections might not be available in the demo.

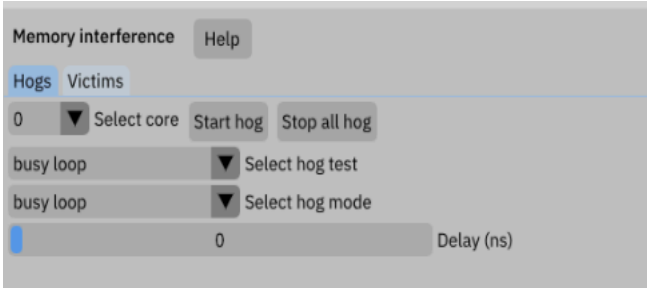
By inserting an IP address in the textbox and clicking on Start, the GUI will load and present a window like this:



On the left bar there are the “interaction buttons”. These buttons are:

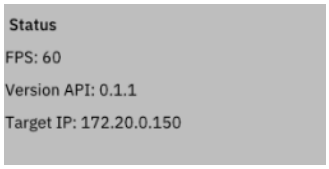
- Regulation:
    - MemPol: You can assign a maximum available bandwidth expressed in MB/s to the cores. This will decrease or increase the number of memory accesses and the performance of the processes scheduled on that specific core.
    - MemGuard: You can enable or disable MemGuard on the cores (note: this feature may not be available in your specific version).



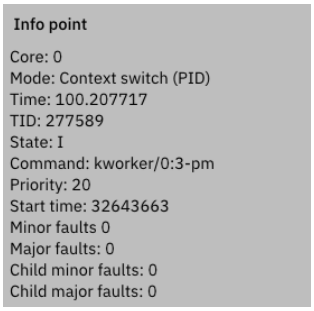


- Interference:
  (Note: this feature is not available in the demo version).
  - Hogs: You can start and stop some memory interference on the cores to observe the behavior of an application under stress. There are different tests and modes that you can choose.
  - Victims: Victim threads simulate periodic real-time threads with a customizable load pattern.

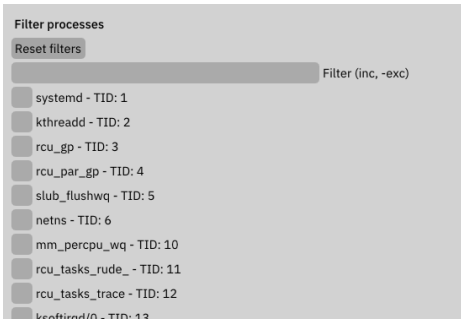
- Info status: Under this tab, you will find information about the target, such as the version of the application or the IP address.



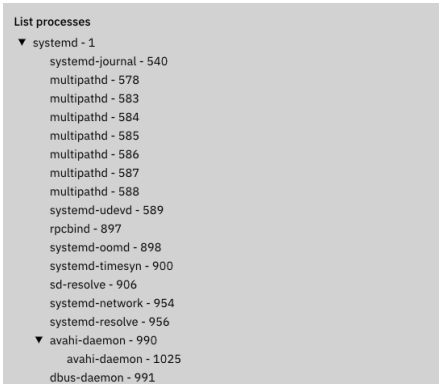
- Info point: Under this tab, when you stop live tracing, you will see additional information about the point on the plot that you are hovering over with the mouse.



- Filter processes: You can filter the thread IDs that you want to see in the context-switch plots.



- List processes: You can view all the threads executing on the target with a tree visualization that lets you identify the parent-children relationships among threads.



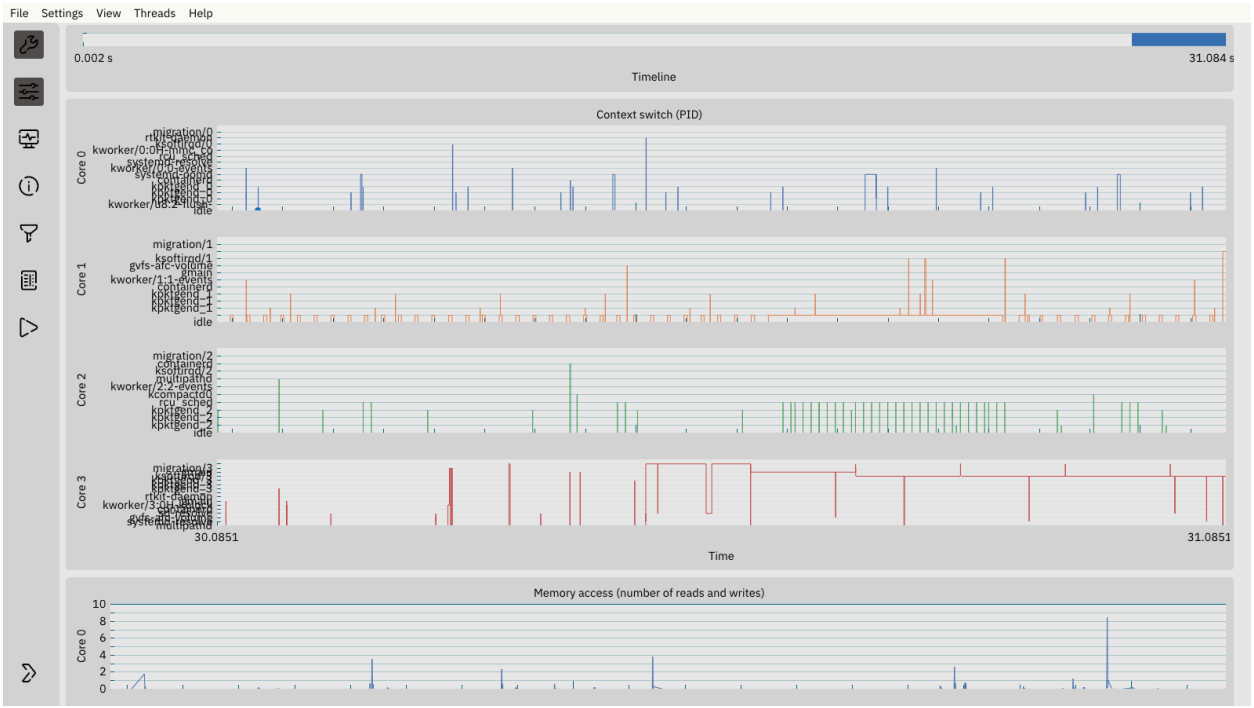
- Play/Stop trace: Start and stop live tracing.
- Open/Collapse menu: You can quickly close the menu with this button.

At the top of the screen, the following menu buttons are also present:

- File: Lets you save sessions to a file and load sessions from a file (note: this feature is not available in the demo version).
- Settings: Lets you enable/disable VSync and switch between light and dark mode.
- View: Lets you show/hide the plots, show/hide the labels on the context switches plots and set a max value for the memory access plots.
- Threads: Lets you change some options (pin to a cpu, set priority, etc.) of the target's threads (main target thread, interference threads, and victims threads).
- Help: Lists all the possible shortcuts and how to handle the plots in offline mode (scroll, zoom, etc.).

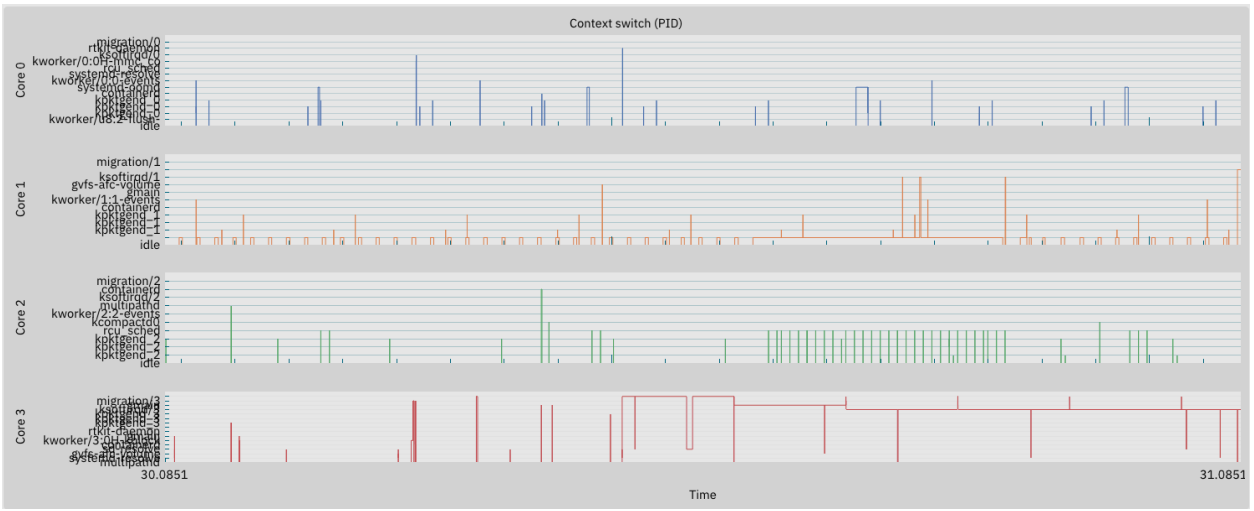
3.3.1 - Live tracing

When you start the live tracing, the plots will begin to update themselves with the data received from the target.

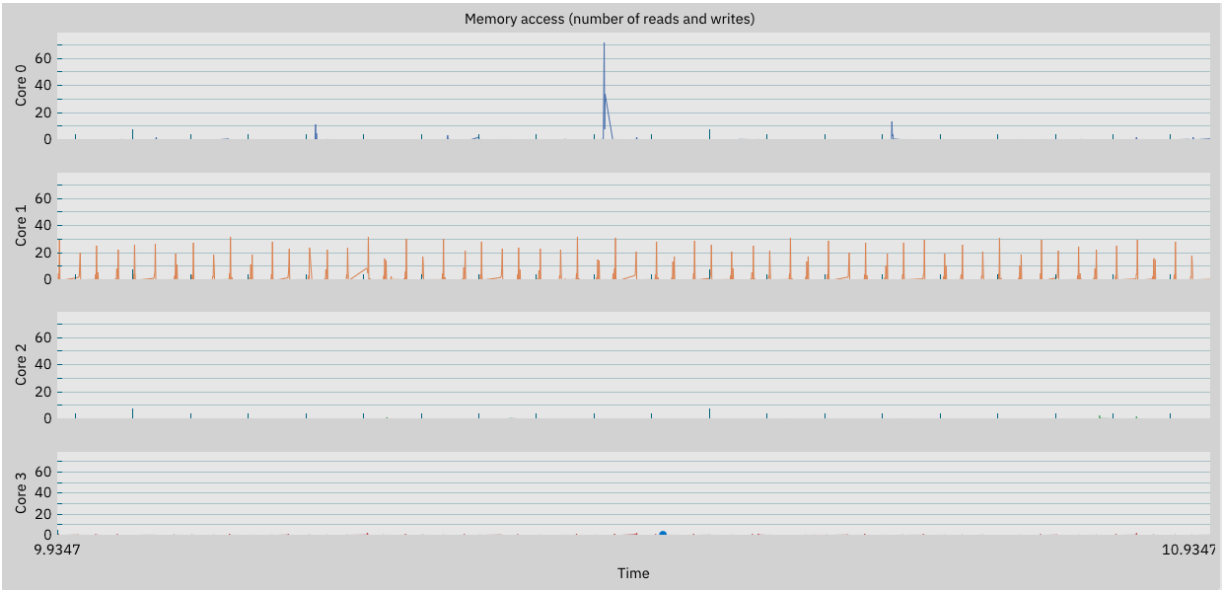


Depending on the available modules on the target, you can see up to three different plots: context switches, number of memory accesses, and memory accesses via perf measures.

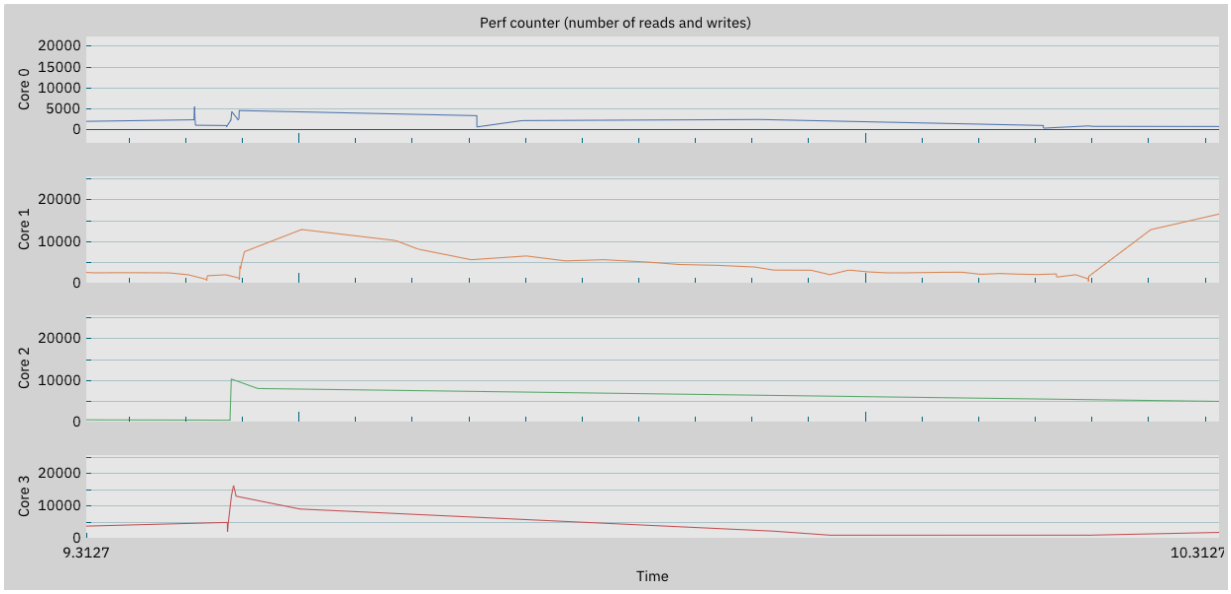
In the context-switch plots area, you will see the last 15 scheduled threads on each core. Note: this information is not available in the demo version.



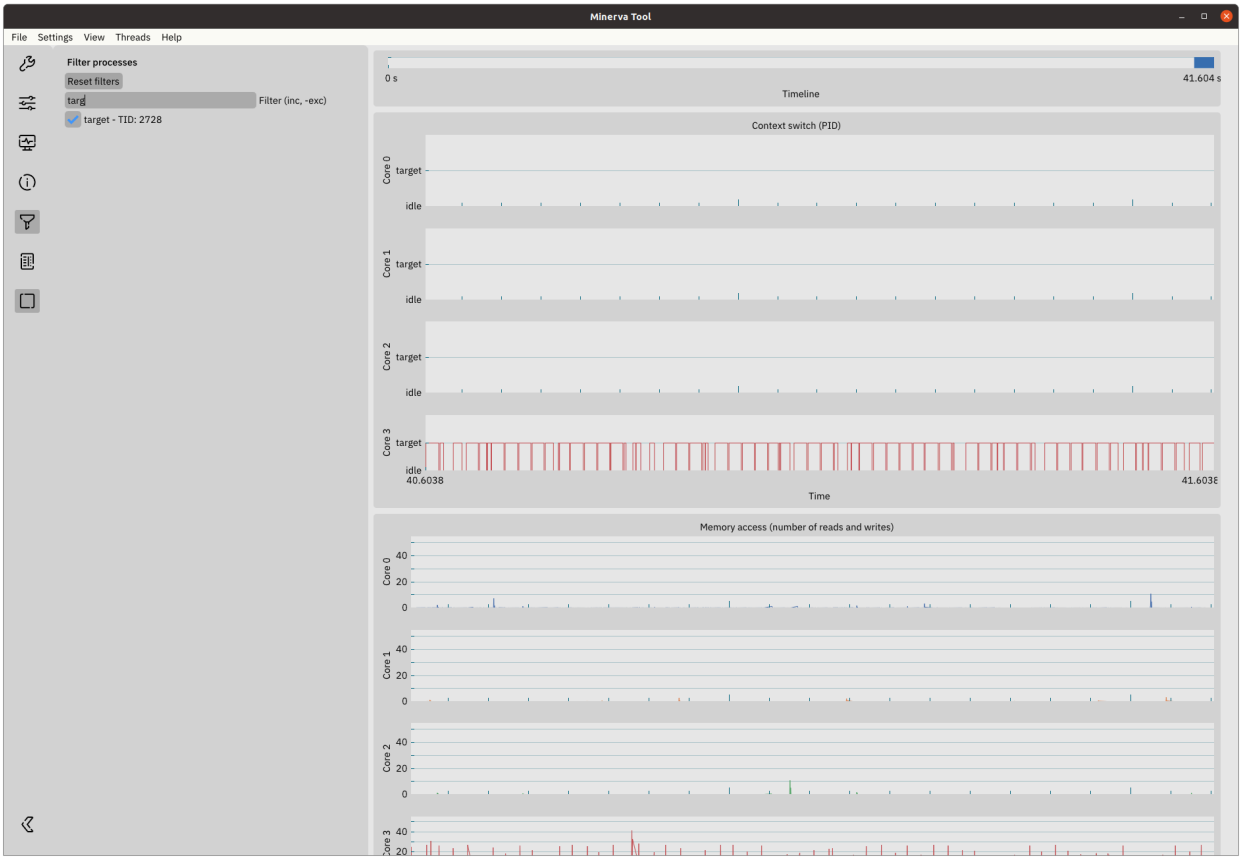
In the Membw plots, you will observe the number of memory accesses (reads + writes) to the L2 cache of each core within a very quick poll interval (for example, on the Kria™ SOM the polling rate is 6.25 µs).



If MemPol is unavailable, you won't see the Membw plots; instead, you will see the Perf plots. Perf measures the same counters (reads + writes of the L2 cache) at a slower polling rate. Note: these plot are not available in the demo version.



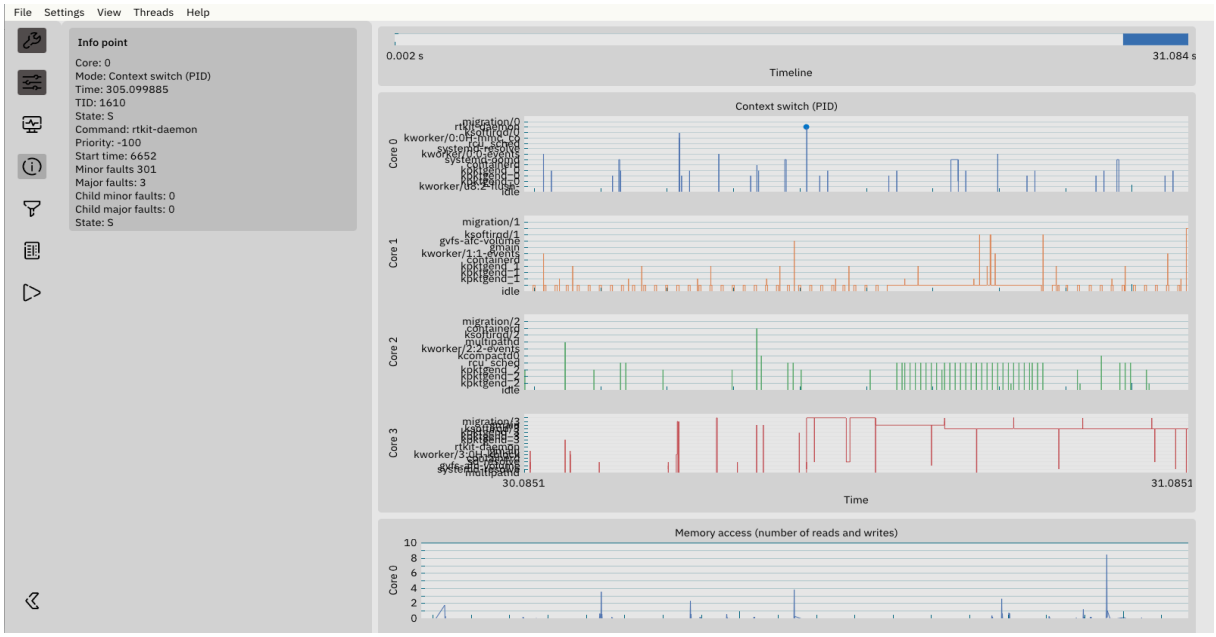
If you want to focus on some specific thread/process, you can filter the context switches plots by selecting the threads that you want to visualize. This will show you only when those specific threads are scheduled.



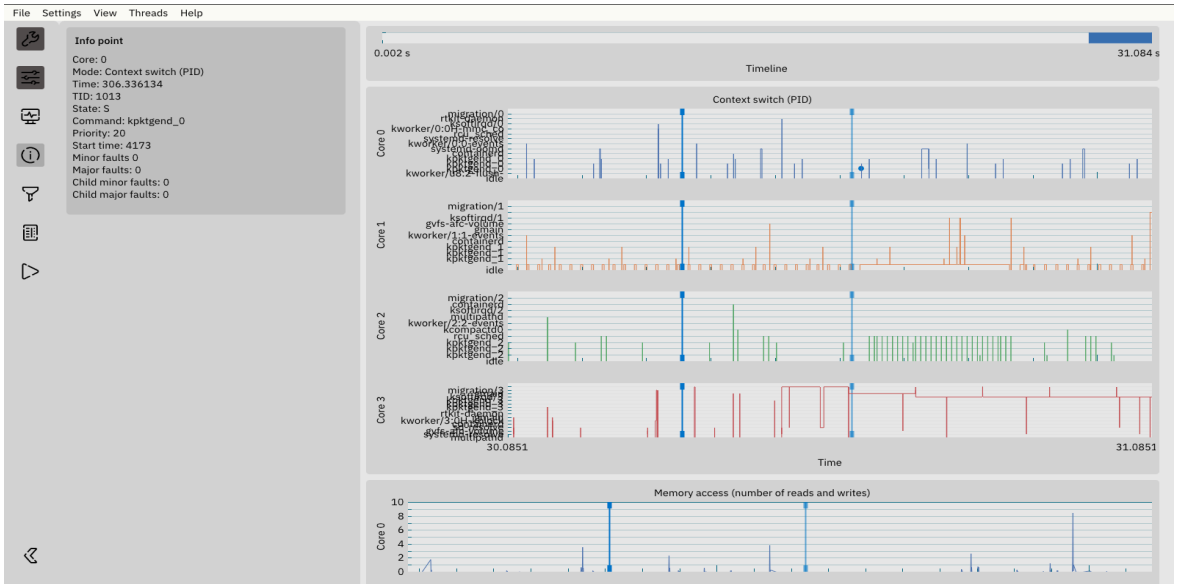
3.3.2 - Offline tracing

When you stop the live tracing, you can view and navigate through the plots. You can zoom in and out, or scroll backward and forward. The timeline shows your position relative to the available data, indicating whether there is data available before or after what you are currently viewing.

By hovering over a point on the context switch plot with the mouse, you can access more detailed information about the process, such as PID, process name, status, priority, and more. Similarly, on the memory access plot, you can obtain the precise number of memory accesses at a specific instant by hovering over the corresponding data point.



If you want a precise way to correlate more plots and, for instance, see which process was running when a particular memory access spike occurred, you can draw a vertical line using the right mouse click.



4 - Examples - Use Cases

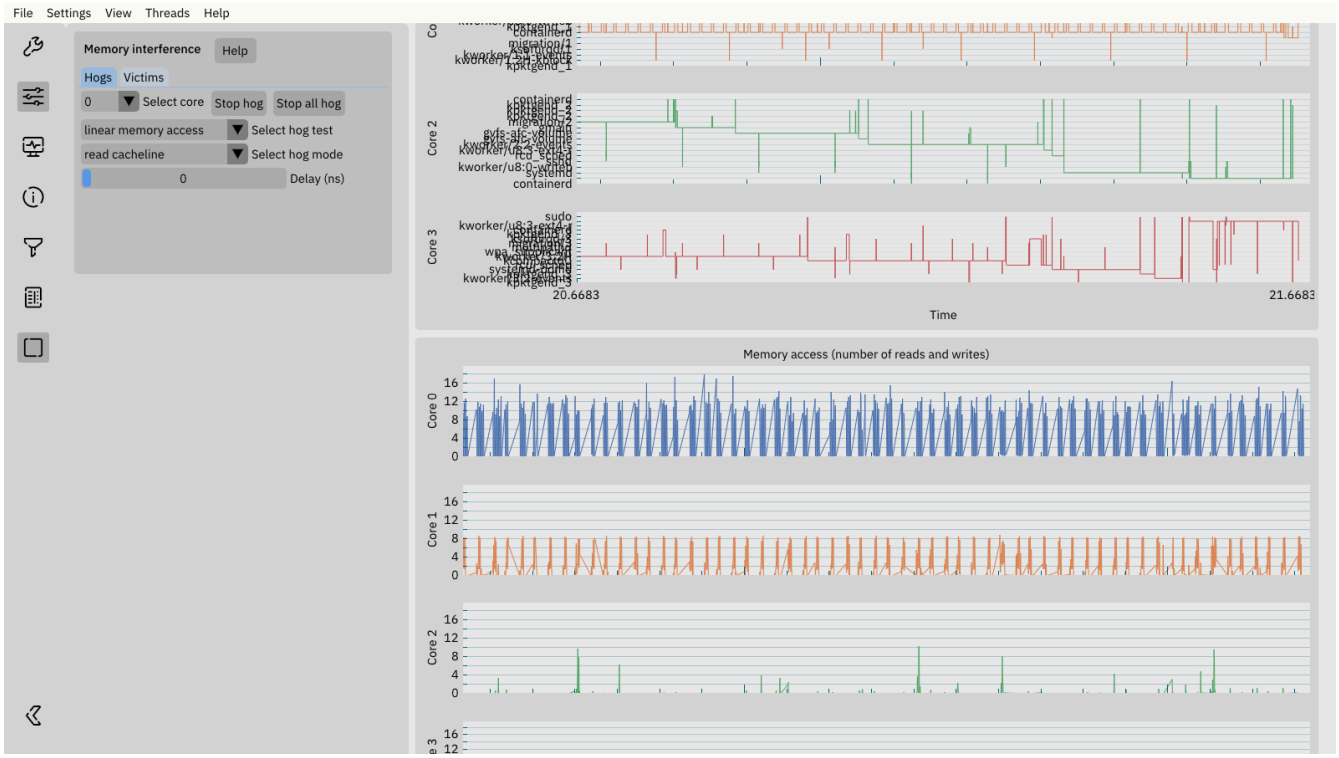
Interference

This scenario addresses two key objectives for our users:

1. **Stress Testing:** Users can push their systems to the limits, intentionally seeking potential breaking points. This helps identify weaknesses and vulnerabilities, allowing for proactive improvements to enhance overall system resilience.
2. **Interference Analysis:** Users can validate and experiment with hypotheses related to interference, particularly bandwidth issues. By creating worst-case scenarios and variations, users gain insights into system behavior under different conditions. This process aids in fine-tuning the system to operate optimally, even in the presence of interference.

Under the Interference tab, in the Hog section, you can replicate some useful memory interference patterns. There are various interference patterns available, which can be used to test interesting corner cases, such as *linear memory access* or *L2 cache thrashing*.

To initiate a hog, simply select the core, the hog test, the hog mode and then click Start Hog. You will immediately observe a change in the number of memory accesses, as shown in the picture below, where a *linear memory access* hog with *read cacheline* mode has been started on core 0.

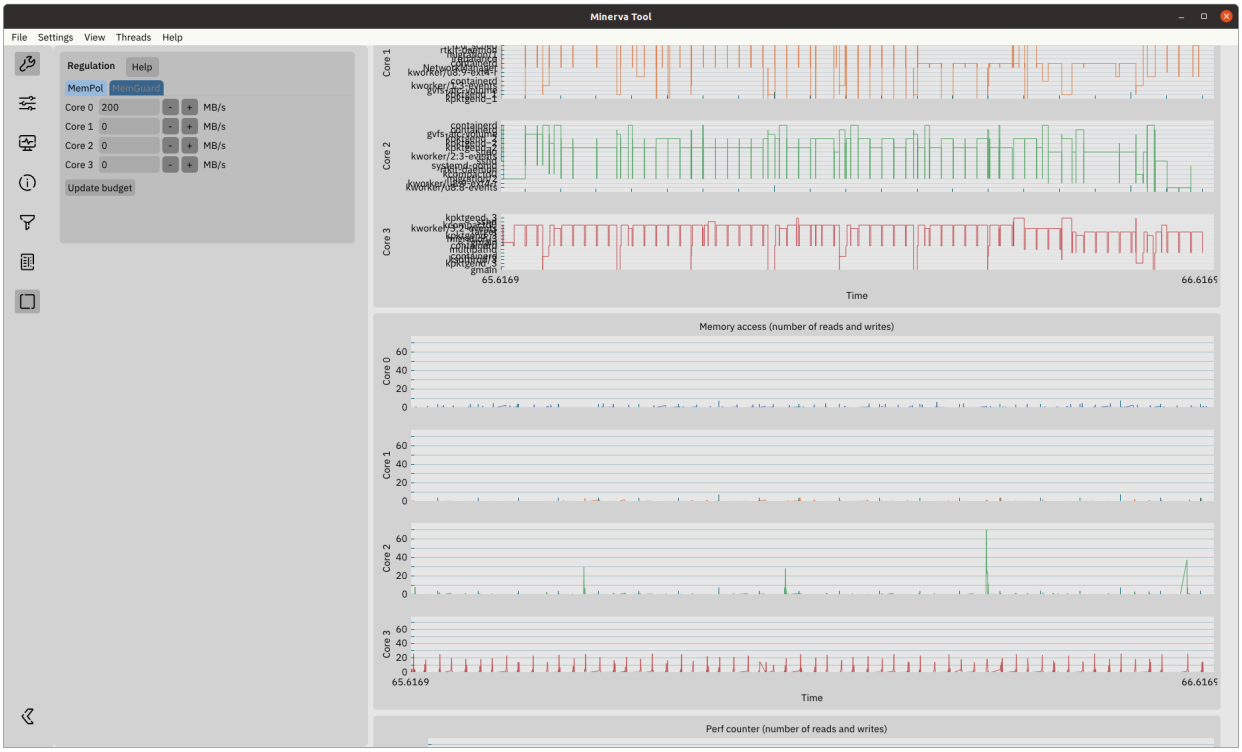


Regulation

1. **Interference Analysis through Control Experiment:** Users can conduct interference analysis by regulating specific tasks. If there's suspicion that interference from certain tasks is causing issues, users can perform a controlled experiment by regulating those tasks. This helps confirm if the problematic situation is indeed linked to interference, providing clarity for further optimization.
2. **Tuning Regulation Configuration:** Users can optimize their system by fine-tuning regulation configurations. This involves determining the highest acceptable regulation on non-critical tasks, while preserving Quality of Service (QoS) and determinism on critical tasks. This use case ensures that the system operates at its optimal performance, striking the right balance between regulation and maintaining the desired level of QoS and determinism.

Under the Regulation tab, you can control the behavior of the cores and restrict their access to cache memory. In the MemPol section, you have the option to cap the maximum available bandwidth of a core by selecting the maximum value in MB/s.

For instance, the image below illustrates what you'll observe when limiting core 0 to 200 MB/s. It is evident that the number of memory accesses for core 0 is significantly lower compared to the other cores.



5 - Getting Help

For any technical issues or inquiries, please feel free to reach out to us at https://www.minervasys.tech/contact.
We are here to assist you and provide the information you need.