SYCLOPS

Deliverable 2.2 – Cross-architecture performance modeling and profiling tools

GRANT AGREEMENT NUMBER: 101092877

Project acronym:	**SYCLOPS**
Project full title:	**Scaling extreme analYtics with Cross architecture acceLeration based on OPen Standards**
Call identifier:	**HORIZON-CL4-2022-DATA-01-05**
Type of action:	**RIA**
Start date:	**01/01/2023**
End date:	**31/12/2025**
Grant agreement no:	**101092877**

D2.2 – Cross-architecture performance modeling and profiling tools

Executive Summary: This deliverable outlines the work done on performance profiling and monitoring tools in "Task 2.2: Benchmark specification and performance profiling" in WP2 of the SYCLOPS project. The SYCLOPS project has developed two key performance profiling and analysis tools—the CARM Tool (developed by INESC) and Adaptyst (developed by CERN).

WP:	2		
Author(s):	Aleksander Ilic, Maksymilian Graczyk		
Editor:	Raja Appuswamy		
Leading Partner:	INESC		
Participating Partners:	CERN		
Version:	1.0	**Status:**	Draft
Deliverable Type:	R	**Dissemination Level:**	PU
Official Submission Date:	06-Oct-2025	**Actual Submission Date:**	30-Sep-2025

Disclaimer

The SYCLOPS consortium consists of the following partners:

No.	Partner Organisation Name	Partner Organisation Short Name	Country
1	EURECOM	EUR	**FR**
2	INESC ID - INSTITUTO DE ENGENHARIADE SISTEMAS E COMPUTADORES, INVESTIGACAO E DESENVOLVIMENTO EM LISBOA	INESC	**PT**
3	RUPRECHT-KARLS-UNIVERSITAET HEIDELBERG	UHEI	**DE**
4	ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE	CERN	**CH**
5	HIRO MICRODATACENTERS B.V.	HIRO	NL
6	ACCELOM	ACC	FR
7	CODASIP S R O	CSIP	CZ
8	CODEPLAY SOFTWARE LIMITED	CPLAY	UK

Document Revision History

Version	Description	Contributions
0.1	Structure and outline	EUR
0.2	CARM tool contribution	INESC
0.3	Adaptyst contribution	CERN
1.0	Final draft	EUR

Authors

Author	Partner
Aleksander Ilic	INESC
Maksymilian Graczyk	CERN

Reviewers

Name	Organisation
Vincent Heuveline	UHEI
Danilo Piparo	CERN
Nimisha Chaturvedi	ACC
Martin Bozek	CSIP

Statement of Originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of Contents

Executive Summary

This deliverable outlines the work done on performance profiling and monitoring tools in “Task 2.2: Benchmark specification and performance profiling” in WP2 of the SYCLOPS project. The SYCLOPS project has developed two key performance profiling and analysis tools—the CARM Tool (developed by INESC) and Adaptyst (developed by CERN).

With the CARM tool, we have developed portable microbenchmarks to generate the CARM model for mainstream CPU architectures, including Intel, AMD, ARM, and especially RISC-V processors that are used in SYCLOPS. We have integrated performance counter (PMU) and DBI analysis of applications to examine performance within the scope of the CARM model. All these features have been implemented into a single solution, providing a one-stop shop for CARM-related analysis, complete with a Graphical User Interface (GUI) to facilitate usage and visualization of results.

With Adaptyst, we have developed an open-source and architecture-agnostic performance analysis tool designed to be portable across programming languages, including C++ with SYCL. Adaptyst supports sampling of both on-CPU and off-CPU activity for all threads and processes of a given program. It can produce data for rendering interactive non-time-ordered and time-ordered flame graphs using a separate tool called Adaptyst Analyser. It also supports custom "perf" events (like performance counters) for analyzing low-level software-hardware interactions. It has been extensively tested on the x86-64, arm64, and RISC-V instruction set architectures available in SYCLOPS EMDC and beyond.

Both these tools are available as open source software in their respective Github repositories mentioned in this deliverable. Substantial dissemination work has also been undertaken to promote their uptake.

1 Introduction

Figure 1 shows the SYCLOPS hardware-software stack consists of three layers: (i) infrastructure layer, (ii) platform layer, and (iii) application libraries and tools layer.

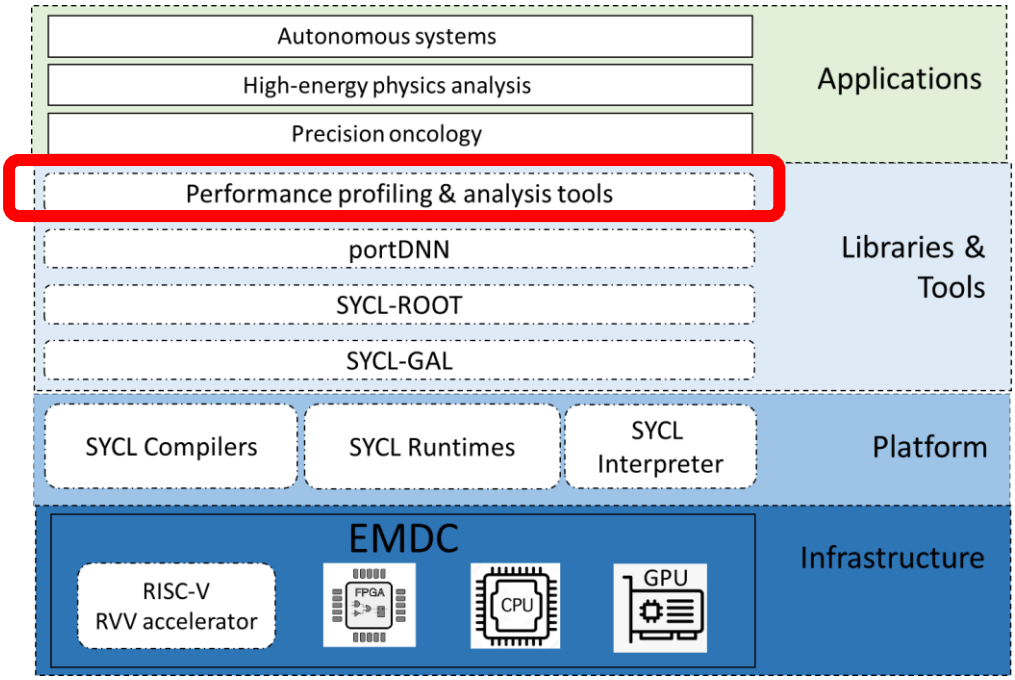

Figure 1. SYCLOPS architecture

Infrastructure layer: The SYCLOPS infrastructure layer is the bottom-most layer of the stack and provides heterogeneous hardware with a wide range of accelerators from several vendors.

Platform layer: The second layer from the bottom, the platform layer, provides the software required to compile, execute, and interpret SYCL applications over processors in the infrastructure layer. SYCLOPS will contain oneAPI DPC++ compiler from CPLAY, and AdaptiveCpp from UHEI. In terms of SYCL interpreters, SYCLOPS will contain Cling from CERN.

Application libraries and tools layer: While the platform layer described above enables direct programming in SYCL, the libraries layer enables API-based programming by providing pre-designed, tuned libraries for various deep learning methods for the PointNet autonomous systems use case (SYCL-DNN), mathematical operators for scalable HEP analysis (SYCL-ROOT), and data parallel algorithms for scalable genomic analysis (SYCL-GAL).

This deliverable presents the work carried out in performance profiling and analysis tools as highlighted in Figure 1.1 in the context of “Task 2.2: Benchmark specification and performance profiling” of the SYCLOPS project. This document provides a high-level overview of the work done on the two performance profiling tools that were developed in the SYCLOPS project, namely, the CARM tool by INESC and Adaptyst by CERN.

This deliverable is structured as follows. Section 1 of this deliverable provides a high-level overview of the overall SYCLOPS architecture and positions this deliverable with respect to both components in the SYCLOPS stack and WP/tasks in the work plan. Section 2 provides an overview of the work done on the CARM tool. Section 3 provides an overview of Adaptyst. Section 4 describes the work on “integrating” the two orthogonal pieces of work, demonstrating how both tools benefited each other to provide a holistic, unified framework for profiling software developed in SYCLOPS and beyond.

2 CARM Tool

To tackle the objectives of in the SYCLOPS project, especially the ones envisioned with the WP2, Task 2.2, the INESC-ID partner has focused on investigating the performance models, profiling and instrumentation tools. To this respect, the roofline model is of particular interest[1] - a performance model known for its easy-to-understand guidelines and useful insights into what bottlenecks are constraining application performance on a given system, such as the Original Roofline Model (ORM)[1], and the Cache-Aware Roofline Model (CARM)[2].

These types of models have varying levels of difficulty in their implementation to be generated for a given system. For example, a fully-compliant ORM implementation requires measuring the memory traffic between cache levels, usually done via cache simulation, which is very time-consuming and architecture-dependent. On the other hand, the CARM requisites are less complex to be made portable across various architectures, since the CARM relies on microbenchmarking data for various ISAs. In fact, the CARM is a widely used roofline model due to its ability to provide a more detailed look at an architecture by considering the diverse characteristics of the memory subsystem, which typically varies across different levels in terms of size, bandwidth, and latency.

Currently, the CARM is supported for Intel architectures via Intel Advisor, while the ORM has rudimentary support in AMD via AMD uProf. These are closed-source tools, while open-source tools like the Empirical Roofline Tool (ERT) are also available but their support is mostly focused on x86-64 architectures, and they do not provide accurate benchmarking mechanisms for architecture maximum performance exploration. The developments in the SYCLOPS project reported in this deliverable aim to close this gap in availability for different architectures such as AARCH64 and RISCV64 by developing an easy-to-use open-source portable tool, that is able to generate the CARM for various architectures based on automatically generated tailored assembly microbenchmarks and also provide application analysis in the context of CARM, with a Graphical User Interface (GUI) to further facilitate the visualization of results.

The main contributions of this CARM Tool are as follows:

- Establishing a cross-architecture microbenchmark methodology;
- The development of portable microbenchmarks to generate the CARM model for various mainstream CPU architectures, such as the most recent Intel, AMD, ARM, and RISC-V processors;
- The integration of performance counter and DBI analysis of applications to then examine in the scope of the CARM model;
- The implementation of these various features into a single tool, providing a one-stop shop for CARM-related analysis, with a GUI to facilitate usage;
- Analysis of different architectures and applications using said tool, with promising results accurately reaching architectural limits.

These contributions are publicly available at https://github.com/champ-hub/carm-roofline and have been disseminated in the following scientific publications:

- Morgado, J., Sousa, L. and Ilic, A.. CARM Tool: Cache-Aware Roofline Model Automatic Benchmarking and Application Analysis. In 2024 IEEE International Symposium on Workload Characterization (IISWC) (pp. 68-81). IEEE, 2024

[1]S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," Communications of the ACM, vol. 52, no. 4, pp. 65–76, 2009.
[2]A. Ilic, F. Pratas, and L. Sousa, "Cache-aware roofline model: Upgrading the loft," IEEE Computer Architecture Letters, vol. 13, no. 1, pp. 21–24, 2013.

- Kabadzhov, I. D., Morgado, J., Ilic, A., and Appuswamy, R. Open, cross-architecture acceleration of data analytics with SYCL and RISC-V. In Proceedings of the 23rd International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms, collocated with Euro-Par, August 26, 2025, Dresden, Germany.
- Rodrigues, A., Sousa, L. and Ilic, A. "A Performance Modelling-Driven Approach to Hardware Resource Scaling". In European Conference on Parallel Processing (pp. 143-154), Cham: Springer Nature Switzerland, 2023

2.1 Background and related work

The roofline model[1] has become a widely used performance model in HPC, correlating system performance with Arithmetic Intensity (AI). In particular, the Cache-Aware Roofline (CARM)[2] evaluates all load/store operations across the entire memory hierarchy in a single plot, providing a true application AI and allowing insight into application-specific load/store ratios, port utilization, ISA precision, and data widths. Its single-plot nature maintains the roofline's simplicity while improving accuracy and usability, as evidenced by its full integration into Intel Advisor.

CARM analysis categorizes applications into memory-bound, compute-bound, or mixed regions. Applications in the memory-bound region benefit from optimizing memory accesses, while compute-bound workloads require better arithmetic unit utilization. Mixed workloads may benefit from both. To construct these roofs, assembly-level microbenchmarks measure sustainable bandwidth at each memory level and peak FP performance. These microbenchmarks stress cache hierarchies and arithmetic pipelines, with results validated using performance counters. From these, performance limits are expressed as:

$$Fa{=}min(Fp, BLx \rightarrow C \times AI),$$

where Fp is peak FP performance and $BLx \rightarrow C$ the sustainable bandwidth of memory level x.

Architectural differences strongly impact these values. For instance, Intel Skylake-X achieves 192 B/cycle L1 bandwidth with AVX-512, while AMD Zen 3 sustains 96 B/cycle with AVX2. ARM Vulcan and RISC-V C920, with fewer load/store units and narrower SIMD (NEON, RVV), sustain only 32 B/cycle. All include two FMA-capable FP units, doubling peak throughput with each wider SIMD extension. However, vendor-reported "peak" metrics rarely align with sustained real-world performance, reinforcing the need for empirical benchmarking.

2.2 CARM Tool: High-level overview

The proposed CARM tool comprises a set of independent modules to provide a complete CARM-based profiling ecosystem, as shown in Figure 2. The tool offers both a command-line and a graphical user interface (GUI) for interaction, allowing users to access stored results from benchmarking and application analysis. Whether activated through the command line or the GUI, the automatic benchmarking and application analysis modules execute user-specified tasks and automatically save the results, which can be visualized within the GUI or as SVG graphs.

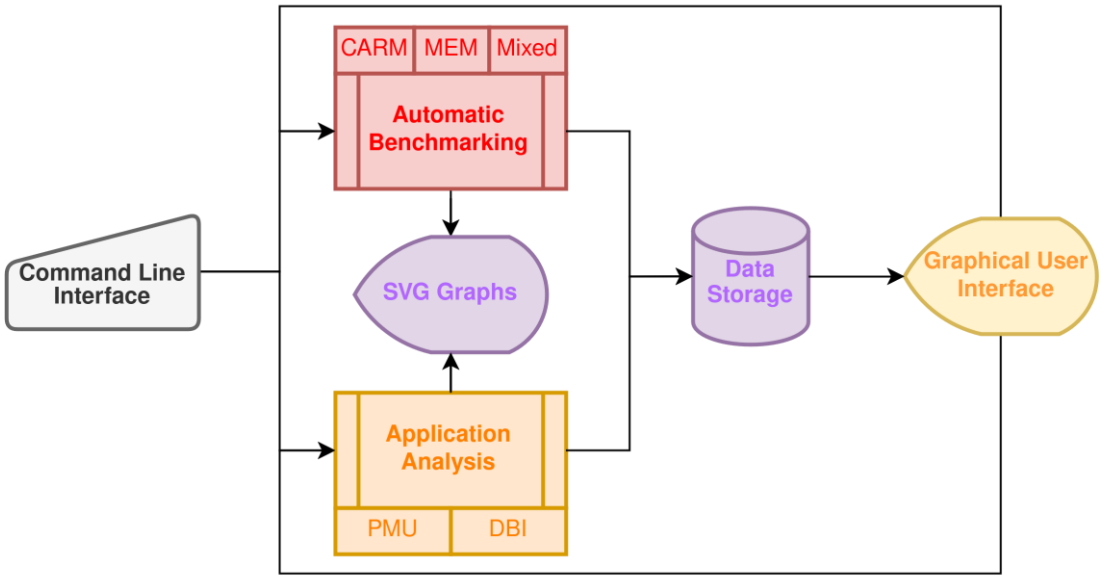

Figure 2. CARM Tool Modules

2.2.1 Automatic Benchmarking

The Automatic Benchmarking module in the CARM tool is managed through a Python script that coordinates automatic benchmark generation and execution. This module conducts various benchmarks to assess memory bandwidth and peak FP performance using automatically generated, tailored assembly-level microbenchmarks across multiple supported microarchitectures. Benchmarks include roofline, multi-level memory bandwidth discovery (referred to as memory curve benchmarks), and mixed-instruction types, detailed via microbenchmarks tailored to specific user-selected options and target specifications provided to the Python script via arguments.

The **--test** argument specifies the benchmark type, such as roofline for comprehensive CARM results, or specific cache levels (L1, L2, L3, DRAM) and FP for focused tests. The **MEM** option triggers memory curve benchmarks, while **mixedL1**, **mixedL2**, **mixedL3**, and **mixedDRAM** target mixed benchmarks, which interleave FP operations with memory accesses at specific memory levels. The **--ISA** argument selects ISA extensions for analysis, offering scalar and vector options like SSE, AVX2, AVX512 (x86-64), Neon (ARM), and RVV0.7/1.0 (RISC-V). The default auto option detects and benchmarks available ISAs. Data precision can be toggled between double and single precision using the **--precision** argument, and the **--threads** argument specifies thread counts for multi-core benchmark execution. The **--ld_st_ratio** argument configures the ratio of load to store instructions, while the **--only_ld** and **--only_st** flags create benchmarks with only load or only store instructions, enabling detailed bandwidth comparisons.

a) *Roofline Benchmarks*: Roofline benchmarks focus on memory levels and two FP benchmarks. The first FP benchmark can be configured to use addition, division, or multiplication instructions via the **--inst** argument, while the second always uses FMA. For memory benchmarks, cache sizes are automatically detected on x86-64 systems using the cpuid instruction, while they can be manually specified for AARCH64 or RISCV64 via a config file or command-line arguments. Results are stored in CSV files, including memory bandwidth (in GB/s and IPC) and peak FP performance (in GFLOPS and IPC). These can be visualized in the GUI (see Figure 3).

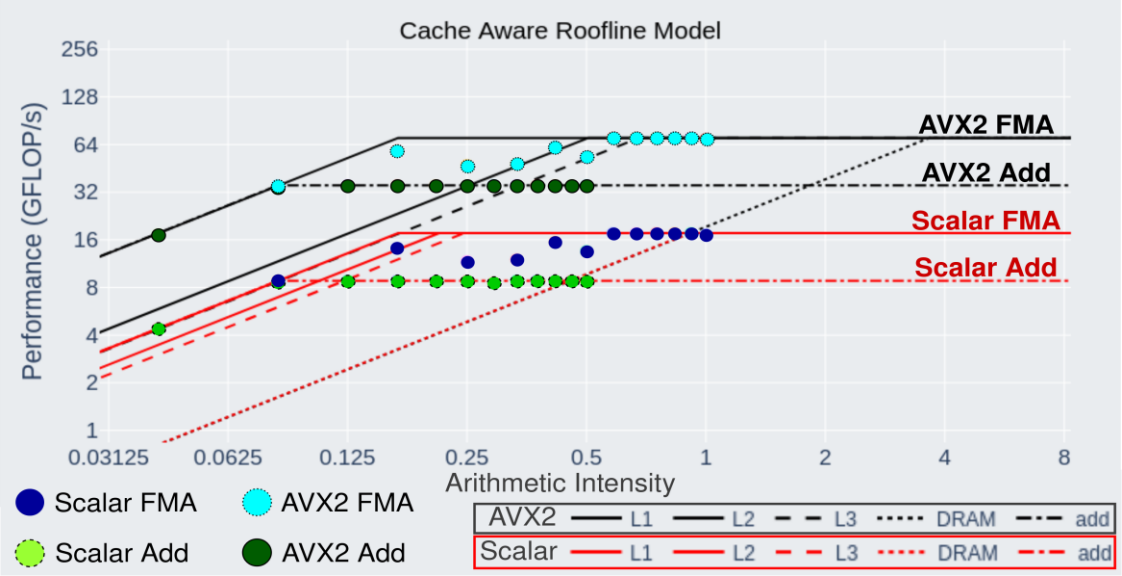

Figure 3. CARM mixed benchmark results and GUI

b) *Memory Curve and Mixed Benchmarks*: Memory curve benchmarks analyze bandwidth across a wide range of problem sizes (2 KB to 512 MB), measuring bandwidth and IPC variations. Results are stored in CSV files and visualized as SVG graphs, with cache sizes optionally included. Mixed benchmarks combine memory and FP operations, targeting specific memory levels. Users can adjust the FP-to-memory ratio and select FP instruction types. Results include AI and GFLOPS metrics, stored in CSV files and visualized in the GUI as CARM graph points (Figure 3). Together, these benchmarks provide a comprehensive view of system performance.

2.2.2 Application Analysis

The proposed CARM tool supports in-depth application profiling via two subsystems: DBI and PMU.

DBI-based profiling is managed via a custom DynamoRIO client or Intel SDE. DynamoRIO supports dynamic opcode counts on x86-64 and AARCH64 (with early RISC-V support), while SDE is limited to x86-64. Users provide the executable path and select DynamoRIO or SDE, specifying installation paths. DynamoRIO is available via download or source compilation, while SDE is distributed as a binary release.

The tool supports Region of Interest (ROI) profiling, enabled through a header file providing API functions (carm_roi_start(), carm_roi_end()). ROI instrumentation measures execution time and categorizes opcodes by type and ISA, enabling detailed FP and AI calculations. Results are stored in CSV files and visualized in the GUI.

PMU-based profiling leverages the PAPI high-level API (PAPI_hl_region_start(), PAPI_hl_region_end()) to calculate AI and GFLOPS for ROI-based application profiling. Future integration with tools like Likwid or Perf may extend this to full-application profiling. PAPI monitors events such as PAPI_LST_INS (load/store instructions), PAPI_SP_OPS (single precision operations), and PAPI_DP_OPS (double precision operations). Each experiment is repeated three times to avoid multiplexing issues, ensuring accurate results without statistical assumptions. Results are integrated into the GUI, similar to DBI profiling.

2.2.3 Graphical User Interface

While all functionalities are available via the command line, the CARM tool also provides a browser-based GUI (Figure 4). The GUI facilitates the configuration and execution of benchmarks, application analysis, and result visualization. It has two main sections: (i) the main window for result visualization (outlined in red), and (ii) a collapsible sidebar (outlined in blue) offering easy access to CARM features without requiring the command line. This design allows users to benchmark, profile, and visualize results in one place with an intuitive, user-friendly interface.

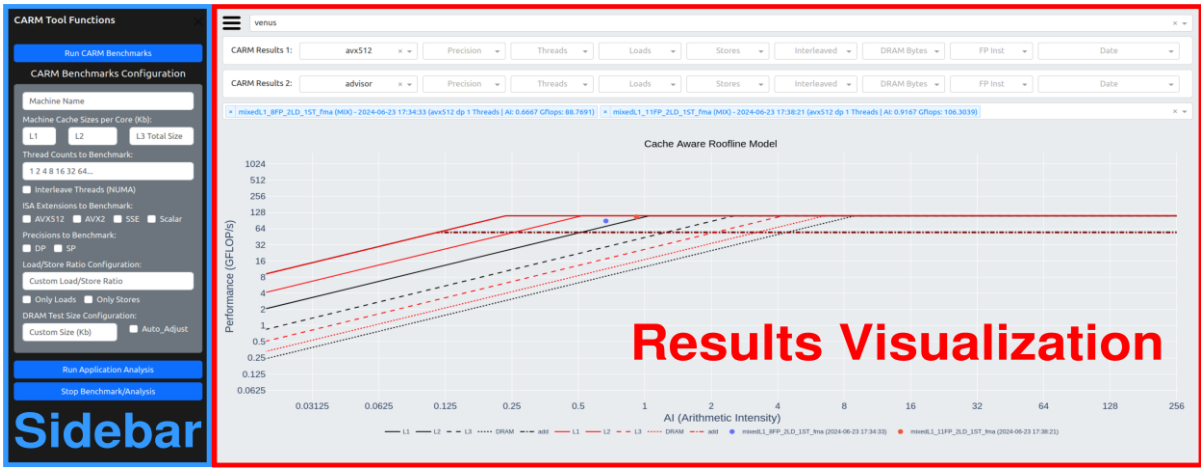

Figure 4. CARM Tool GUI Overview

2.3 Experimental validation

This section presents the experimental results of this work, covering the benchmark outcomes of the developed CARM and memory benchmarks within the CARM tool, as well as their validation through various means, including benchmarks, DBI, and PMU analysis. The various environments used for obtaining these experimental results, including Intel Xeon Gold 6140 (Venus) and 6528R (CSL), AMD Threadripper PRO 5975WX (Cara), ARM Cavium ThunderX2 CN9980 (Armq), RISC-V MilkV Sophon SG2042.

2.3.1 CARM Benchmarking Results

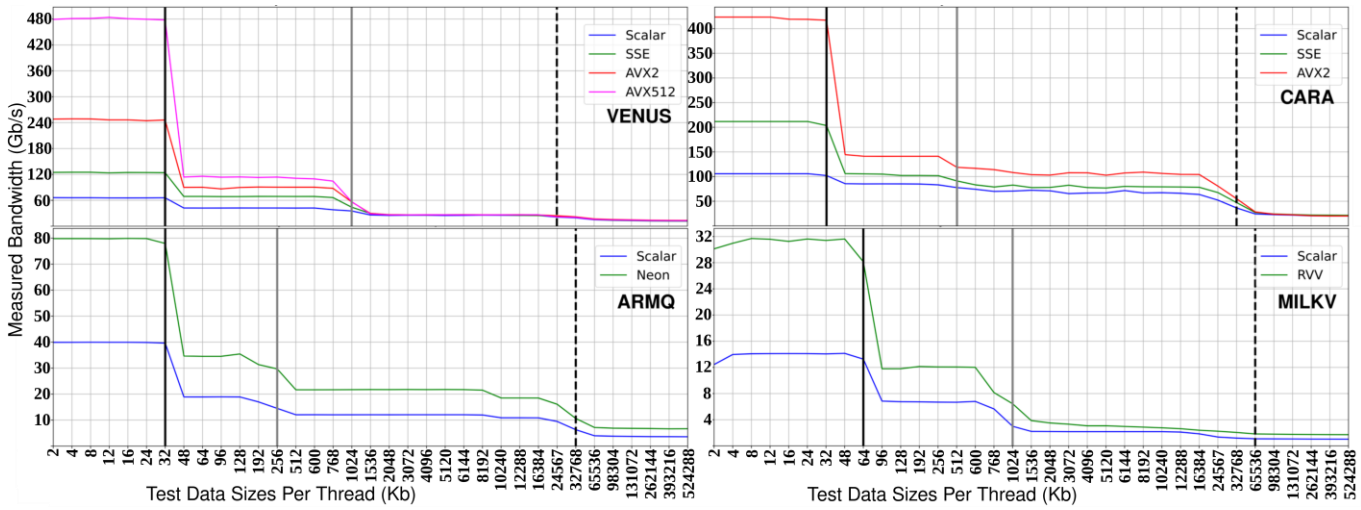

Figure 5. Memory curve benchmark results

x86-64 CPUs: For the x86-64 representation, the Venus and Cara machines were tested. The Venus machine contains an Intel Skylake-X CPU which supports all x86-64 ISA extensions used by the CARM tool, while the Cara machine contains an AMD Zen3 which supports up to AVX2. These CPUs both feature two load and one store units per core, theoretically achieving up to three IPC at the L1 cache level.

In order to better assess the complete performance of their memory subsystem, the memory curve benchmarks were conducted. These tests varied in load/store ratios, thread counts, and ISAs to measure their effects on bandwidth and IPC, as detailed in the memory curve graph for a 2-load-to-1-store ratio on one thread for all available ISAs (Figure 5). From the various load/store ratios tested, it was observed that the two loads per store ratio produced the highest bandwidth and IPC values, matching the CPU's

load/store unit ratio. Load-only and store-only tests produced IPC counts of two and one respectively, which indicates the presence of two load and one store unit per core.

On Venus, however, during the two loads per store test, the maximum theoretical IPC of three was not reached, only achieving a peak of 2.046 IPC. This value closely matches the sustained bandwidth reported in the Intel Optimization Manual, of 2.078 IPC, with only a 1.54% deviation. For the L2 cache, while the manual cites a bandwidth of 0.813 IPC, our load-only benchmarks showed higher IPCs up to 0.964. L3 bandwidth analysis was complicated due to the small 1408 KB L3 slice per core. Tests with a 1216 KB data size, fitting between L2 and L3 limits, recorded a 0.194 IPC, 17% below the reported 0.234 IPC for L3.

Testing on the Cara machine yielded similar results. While there are no official sustained bandwidth values reported by AMD, the memory benchmarks managed to reach three, one, and 0.7 IPC for the L1, L2, and L3 memory levels, which closely match the theoretical architectural limits of the Zen3 architecture, with only the L3 result being 30% below the theoretical maximum.

After analyzing the memory subsystem, we turn to the FP performance of the Skylake-X and Zen3 CPUs, which, with two FP units per core capable of AVX-512 and AVX2 operations respectively, can theoretically reach up to 2 FP IPC. To verify this, the FP CARM benchmarks were executed for both AVX512 and Scalar on Venus. Results showed IPC counts of 1.88 and 1.98 for AVX512 and scalar respectively, indicating that while not perfectly achievable, we are able to get close to the theoretical 2 FP IPC, especially for the scalar instructions. On Cara, two FP IPC were accurately reached, matching the expected theoretical maximums. From this analysis, we can conclude that the CARM benchmarks on the x86-64 architectures are showing promising results, closely following the theoretical limits of the Skylake-X and Zen3 CPUs. These benchmarks will be validated in the next section via various methods, to confirm that these measured values actually correspond to the real execution of the instructions in the assembly microbenchmarks.

AARCH64 CPUs: The ThunderX2 CPU on the Armq machine was also benchmarked. Equipped with the Neon SIMD extension, it has two load/store units capable of executing Neon operations, theoretically achieving up to two memory IPC. To explore these limits, memory curve benchmarks were conducted across various load/store ratios, in a similar fashion to the tests for x86-64. These benchmarks indicated that the load-only ratio performed the best on Armq.

The load-only ratio benchmark reached an IPC of two, closely matching the expected IPC. Using only stores resulted in an IPC of one, suggesting that only one of the load/store units in the CPU is capable of performing store operations. This discrepancy explains the stability of the load-only results, as introducing store instructions appears to cause conflicts within the dual-capable load/store unit. Furthermore, despite the ThunderX2 reportedly having a 64 bytes per cycle bandwidth to the L1 cache, suggesting a theoretical IPC of four, the practical IPC drops below one when accessing the L2 cache and decreases further with L3 cache usage.

The Armq CPU, equipped with two Neon-capable FP units per core, theoretically achieves an IPC of two for FP arithmetic. The FP CARM benchmarks, executed for both Neon and scalar extensions, show measured IPC values closely aligned with the theoretical two IPC, exhibiting an average deviation of 0.6883%.

RISC-V CPUs: The Sophon SG2042 CPU in the MilkV machine contains a reported two load/store units per core, suggesting a theoretical IPC of two. Memory curve tests using the RVV extension indicate a peak IPC around one using a two loads per store ratio, which was the best-performing ratio. This suggests the presence of only one effective load/store unit. Furthermore, store-only benchmarks consistently showed an IPC of about 0.5 across various data sizes, spanning from within the L1 cache limits to most of the L3 cache capacity.

This trend is likely influenced by the C920 core's adaptive write-allocate policy [18], in conjunction with a 16 MB write buffer, which maintains steady bandwidth measurements across most data sizes. For FP performance, the MilkV machine, with two RVV-capable FP units per core, achieves an IPC close to two,

as confirmed by CARM FP benchmarks with a minimal deviation (0.77%) from the theoretical maximum for both RVV and scalar instructions.

2.3.2 CARM Benchmark Validation

The validation of these benchmarks must be conducted to ensure the obtained results are indeed representative of the peak performance of the various machines.

Examining the AMD Zen3 CPU on the Cara machine via mixed benchmarks (Figure 2), which allow users to adjust the FP-to-load/store instruction ratio of benchmarks targeting a specific memory level, provides insight into performance when stressing both memory and FP units.

By using mixed benchmarks, we can visualize the system performance when both subsystems are stressed simultaneously, which should still allow the CPU to reach near the limits set by the CARM benchmarking. The Zen3 CPU, with an optimal ratio of two loads per store, executed various mixed benchmarks ranging from 0.0417 to 0.25 AI for addition and 0.0833 to 0.5 AI for FMA instructions.

Mixed benchmarks targeting the L1 cache incrementally increased the FP instruction ratio to a maximum of 12 per three memory operations, capturing critical performance points around the ridge point of the CARM where FP limitations start to dominate. These benchmarks, executed on the Cara machine using AVX2 and scalar instructions on one thread, showed that the Cara machine closely approached the CARM limits from previous CARM benchmarking, as illustrated in Figure 6, where each dot corresponds to a mixed benchmark execution of a particular AI.

Errors in AVX2 were lower, averaging 13.69% for FMA and 0.16% for addition, compared to scalar benchmarks which showed errors of 13.97% and 1.11% respectively. While errors were considerably greater for FMA instructions in both cases, this suggests that less complex operations like addition maintain performance better under mixed conditions.

3 Adaptyst[3]

Adaptyst is an open-source and architecture-agnostic performance analysis tool designed to be portable across programming languages (including C++ with SYCL). In its current development stage, it runs on Linux, is based on “perf” with the custom patches, and:

- samples both on-CPU and off-CPU activity of all threads and processes of a given program,
- minimises the risk of incorrectly-collected stack traces as long as the program is compiled with frame pointers (this is done by detecting improper CPU and kernel configurations),
- produces data for rendering interactive non-time-ordered and time-ordered flame graphs in a web browser using a separate tool called Adaptyst Analyser,
- supports custom “perf” events (e.g. performance counters) based on sampling for analysing low-level software-hardware interactions,
- integrates with CARM Tool to provide cache-aware roofline insights into the program (see the next chapter for more details),
- is tested on the x86-64, arm64, and RISC-V instruction set architectures.

Figure 1 presents the general flow of Adaptyst. The user starts the tool by executing the “adaptyst” command, which runs “adaptyst-server” in the background by default (there’s also an option of putting “adaptyst-server” on another machine and connecting to it via TCP) and configures “perf”. Afterwards, profiling events are pre-processed by special Adaptyst Python scripts in real-time with the help of “perf” and streamed to “adaptyst-server” (via file descriptors or TCP), which carries out proper processing, e.g. producing flame graph data and saving everything to persistent storage.

Figure 6: Adaptyst flow.

The output format is open and non-encrypted, it can currently be inspected by checking out the Adaptyst source code. The performance analysis results are usually inspected by a dedicated tool called Adaptyst Analyser: it renders an interactive website showing the timeline view with a tree of threads/processes, each containing on-CPU and off-CPU runtime lengths along with (if available) flame graphs, spawning stack traces, corresponding source codes of a profiled program. In case of having run roofline profiling (with the help of CARM Tool benchmarks), a cache-aware roofline graph can be also displayed, with code-segment-specific points plotted by the user. A screenshot of the sample website is shown in figure 2.

Even though Adaptyst is a code profiler in the current stage, its future roadmap goes beyond profiling: the goal of the project after SYCLOPS is becoming a comprehensive performance analysis and full-stack system design/compilation tool. Specifically, the plan is making Adaptyst generate the most optimal software-hardware solution for any given user workflow from anywhere in the computing spectrum (from embedded to distributed/high-performance computing) while taking into account all sides of computation: software optimisations, hardware customisations, choosing compute units from CPUs, GPUs, FPGAs etc., planning out storage and networking, designing memory hierarchies etc.

[3]The contents of this chapter are based on the Adaptyst website made as part of SYCLOPS: https://adaptyst.web.cern.ch (access: 24/09/2025)

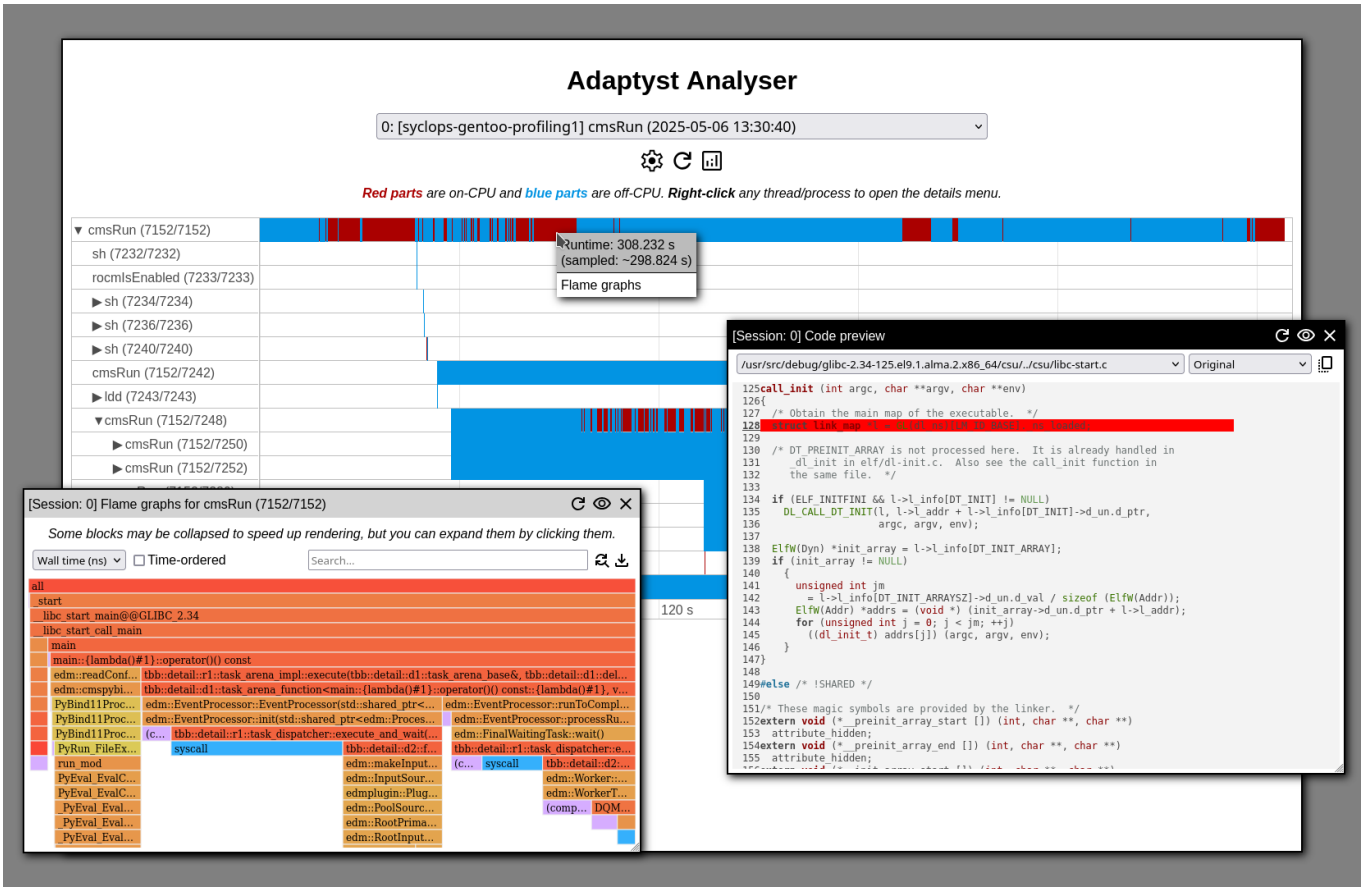

Figure 7: Sample Adaptyst Analyser screenshot.

In the near future, the modular design of Adaptyst will be finalised and published, where functionality related to performance analysis of a system/hardware component like a CPU will be delegated to separate modules that can be developed by anyone. This will allow the project to keep up with the pace of the market development, e.g. by establishing collaboration with hardware companies that would work on the Adaptyst modules for their products.

Adaptyst has been disseminated in several events and through publications:

1. A poster at the ACAT'24 conference on 11-15 March 2024
2. A talk at the Compute & Accelerator Forum at CERN on 8 May 2024
3. A talk at the Open-Source RISC-V Software Workshop co-located with RISC-V Summit Europe 2024 on 28 June 2024
4. A talk and a paper at the CHEP'24 conference on 19-25 October 2024
5. A poster and a paper at the IEEE HPEC 2025 conference on 15-19 September 2025, with the Outstanding Short Paper Award
6. An all-day workshop at CERN dedicated to Adaptyst on 29 September 2025
7. The source code of Adaptyst is available on GitHub: https://github.com/adaptyst/adaptyst. More details about the project can be found at the website: https://adaptyst.web.cern.ch.

4 Integration

Adaptyst and CARM Tool have integrations with each other: the first section explains how Adaptyst can use CARM Tool to provide cache-aware roofline insights into a profiled program and the second section does a similar thing vice versa.

4.1 Adaptyst using CARM Tool

If enabled by the user, Adaptyst integrates with CARM Tool in two complementary ways: it calls CARM Tool directly to obtain cache-aware roofline benchmark data for a specific machine and uses the expertise of the authors of CARM Tool to profile a given program with roofline-specific “perf” events / performance counters in mind.

When performance analysis results are opened in Adaptyst Analyser, it is possible to view various roofline graphs produced by CARM Tool benchmarking. Additionally, the user can plot code segments from thread/process flame graphs as points on the roofline graphs to quickly determine whether these are more memory- or compute-bound. This fine-grained roofline analysis without requiring the user to specify code regions explicitly is a feature which normally neither Adaptyst nor CARM Tool offers on its own. The screenshot of a sample cache-aware roofline graph with code-segment-specific points is presented in figure 3.

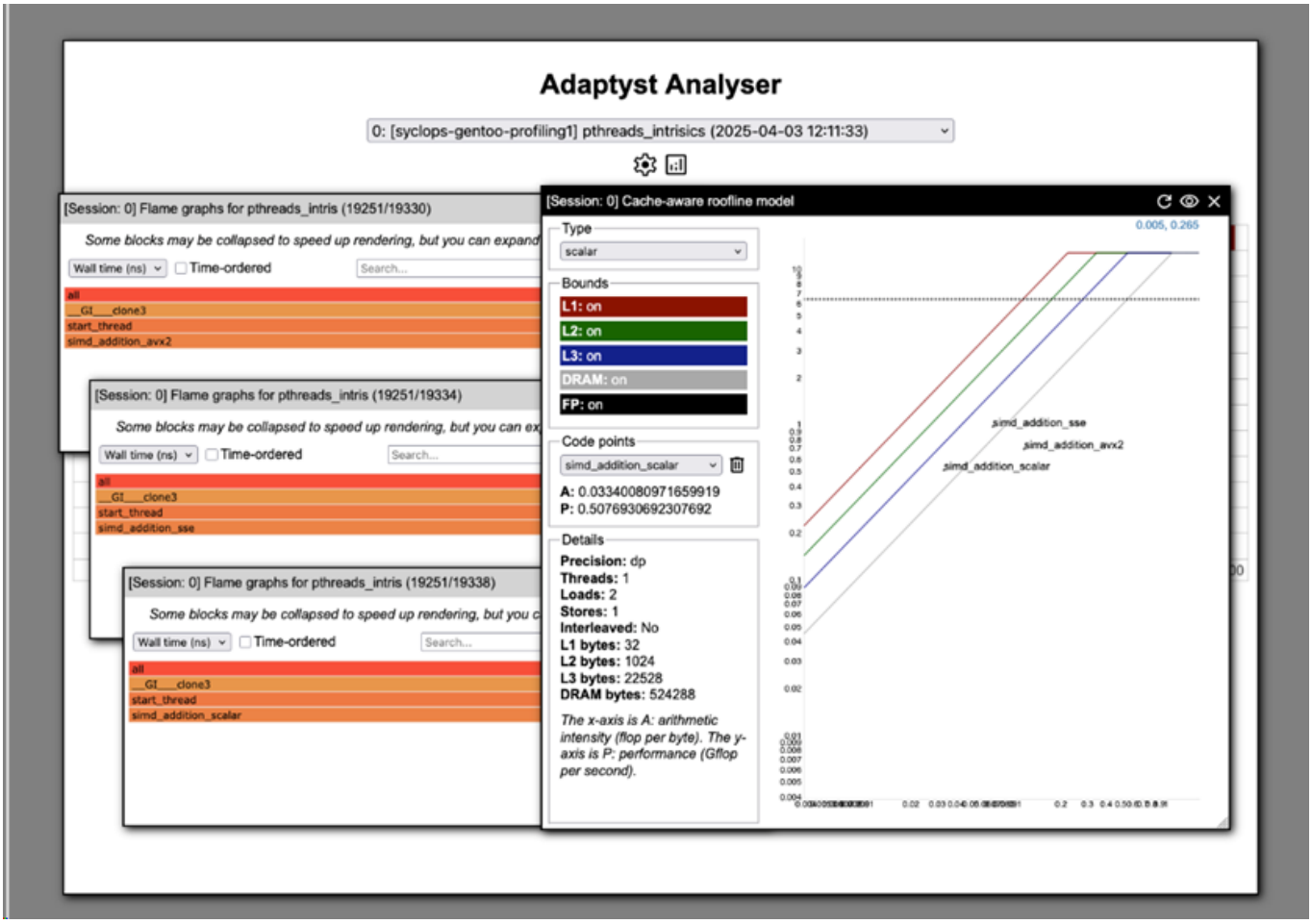

Figure 8: Sample cache-aware roofline graph with three code segments from three different flame graphs plotted as points.

4.2 CARM Tool using Adaptyst

The CARM Tool on the other hand has necessary facilities to read application traces and performance data obtained using Adaptyst, which can be loaded into the CARM Tool GUI. This allows users to view cache-aware roofline models for their system with application behaviour captured by Adaptyst, combining low-level code profiling with CARM's ease of use and insight on application optimization for a given architecture.

Using the CARM GUI, users can visualize roofline graphs generated from the CARM Tools benchmarking and Adaptyst's profiling data. Code segments identified by Adaptyst can be plotted automatically in the CARM, making it possible to assess whether they are compute or memory bound without having to define regions of interest. The CARM Tool can also detected what kind of ISAs are used by an application analyzed using Adaptyst and select the most appropriate CARM results to obtain an accurate analysis of application bottlenecks.

5 Conclusion

This deliverable concludes the work done in Task 2.2 of the SYCLOPS project. Through this work, we have successfully developed and integrated two major performance profiling solutions: the CARM Tool (by INESC) and Adaptyst (by CERN). The CARM Tool provides an open-source, portable platform for generating the Cache-Aware Roofline Model (CARM), addressing the gap in tooling for architectures like AARCH64 and RISCV64 by using cross-architecture microbenchmarking and integrating application analysis via performance counters (PMU) and DBI. Adaptyst complements this by offering an architecture-agnostic performance analysis tool that uses "perf" to sample on-CPU and off-CPU activity, generating interactive non-time-ordered and time-ordered flame graphs.

The most significant achievement is the integration of these two orthogonal pieces of work, resulting in a holistic, unified framework for profiling SYCLOPS software and beyond. This integration enables powerful analysis: Adaptyst calls the CARM Tool to obtain architectural benchmark data, allowing the user to plot specific code segments from flame graphs directly onto the roofline graphs for rapid assessment of whether they are memory-bound or compute-bound. Conversely, the CARM Tool can read performance data generated by Adaptyst, allowing users to visualize application behaviour alongside the CARM model and automatically detect the Instruction Set Architectures (ISAs) used for accurate bottleneck analysis.

All the work done on these tools have already been made publicly available in their respective Github repositories mentioned in this document. In addition to various dissemination efforts listed in this document, we have also published technical blogs on CARM tool and Adaptyst on the SYCLOPS website, and technical talks that can be found in the SYCLOPS Youtube channel.