Research and Development (Upbring and Optimization) of the AERO System-Level Software Stack on the EU Processor

v1.0

Deliverable Number:	D3.1
Due Date:	30.06.2024
Date Of Submission:	10.07.2024
Nature:	OTHER
Dissemination Level:	PU
Work Package:	WP3
Lead Beneficiary	FORTH

Grant Agreement No 101092850

Funded by the European Union

Document Control Sheet

Deliverable Title:	Research and Development (Upbring and Optimization) of the AERO System-Level Software Stack on the EU Processor v1.0
Authors:	Polyvios Pratikakis (FORTH)
Contributors:	Uwe Dolinsky (CPLAY), Michele Paolino (VOSYS), Stefanos Gerangelos (VOSYS), Iakovos Kolokasis (FORTH), Anthony Chazapis (FORTH), Sratos Psomadakis (ICCS), Massimiliano Donati (UNIPI)
Reviewers:	Stefanos Gerangelos (VOSYS), Sergio Saponara (UNIPI)
Approved By:	Christos Kotselidis (UNIMAN), Dionisios Pnevmatikatos (ICCS)

Document History

Version	Date	Status	Description/Comments
0.1	11.06.2024	Draft	Initial version with Table of Contents
0.2	18.06.2024	Draft	First Draft for internal review
0.3	03.07.2024	Draft	Final Draft for internal review
1.0	10.07.2024	Final	Final version submitted to EC

DISCLAIMER

AERO has received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101092850. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the granting authority. Neither the European Union nor the granting authority can be held responsible for them.

TABLE OF CONTENTS

Executive Summary

This document describes results obtained by WP3 during the reporting period (M18). The document is structured to present each component separately, and includes description of each component, work done, results, description of the repository of each component and instructions to build and test each component.

List of Abbreviations & Acronyms

Abbreviation/Acronym	Meaning
AVX	Advanced Vector eXtensions
CTS	Conformance Test Suite
EPI	European Processor Initiative
HE	Homomorphic Encryption
HW	Hardware
OCK	oneAPI Construction Kit
SYCL	Khronos-standardised C++ API for accelerator programming
SVE	Scalable Vector Extensions

1 Introduction

This deliverable entails the intermediate software release (M18), along with the optimized software modules as emerged at the end of the baseline phase. This is work performed within Work Package 3. WP3 is composed of three tasks, namely T3.1 Operating Systems and Virtualization support, T3.2 Firmware and Drivers, and T3.3 Software/Hardware Security extensions.

This document presents work done by all tasks during the reporting period. Namely, with respect to T3.1, Section 2 presents operating system kernel work on Elastic Translations and Section 3 presents VOSySmonitor. With respect to T3.2, Section 4 presents the OpenCL driver for OneAPI Construction Kit. With respect to T3.3, Section 5 presents Homomorphic Encryption and the CKKS Benchmark. Section 6 presents work done on RISC-V virtualization.

2 ARMv8-A Elastic Translations

2.1 Overview

Elastic Translations (ET) are extensions to the Linux kernel memory management subsystem to efficiently exploit the contiguous bit feature of the ARMv8/9-A address translation structures (page tables). ET can assist in alleviating the address translation overhead of cloud workloads running on the ARMv8/9-A architecture of the EU processor.

The modified Linux kernel source code as well as the required userspace tools are hosted on the AERO GitHub organization (https://github.com/AERO-Project-EU/et).

2.2 Hardware & Software Specification of Tested Platforms

Since the Rhea platform is not available, two alternative platforms have been used for developing and testing ET, an Ampere Altra server and an NVIDIA GraceHopper system.

2.2.1 Hardware Specifications

The hardware characteristics of the used platforms are described in Table 1.

Table 1. Hardware specifications of platforms used for ET.

Hardware Specifications		
System	Ampere Altra Mt.Jade	Supermicro ARS-111GL-NHR
CPU	2x Ampere Altra (Q80-30), 160x ARMv8.2-A Neoverse N1 cores	1x NVIDIA Grace, 72x ARMv9 Neoverse V2 cores
Memory	4x 64GiB Samsung DDR4-3200	480GB ECC LPDDR5X
Disk	Samsung 1TB M.2 PCIe 3.0 x4 NVMe SSD	Intel 1.92TB E1.S PCIe 4.0 x4 NVMe SSD

2.2.2 Software Specifications

The software specification of the platforms for using ET are presented in Table 2.

Table 2. Software specifications for ET.

Software Specifications	
OS	Ubuntu 22.04 LTS
Linux Kernel	v5.18.x

2.3 How to build the technology?

Using the provided Linux kernel Kconfig, build and install the Linux kernel:

```
# make Image.gz modules
# make modules_install
# make install
# reboot
```

2.4 How to run?

On an ET-enabled kernel, you can use the provided *prctl* userspace tool to enable ET on a per-process base. For example, you can run the provided *hashjoin* microbenchmark with ET by using:

```
# prctl --ca --et ./hashjoin
```

3 VOSySmonitor

3.1 Overview

VOSySmonitor is the VOSYS firmware product that leverages ARM's TrustZone to isolate workloads running between the Secure and Non-Secure world. In AERO, VOSySmonitor will be ported to the project SoC and will be extended to enable the execution of a secure hypervisor in the Secure World to run strongly secure and isolated virtual machines (Secure VMs). VOSySmonitor is an ISO26262 certified proprietary software, and will be provided to AERO partners for the scope of the project in a binary form.

VOSySmonitor porting to Neoverse V1 is ongoing. Nevertheless, the firmware already boots on Fastmodels FVP platform. The current version of VOSySmonitor is provided in the form of a binary in the AERO GitHub repository space (https://github.com/AERO-Project-EU/VOSySmonitor). In the following sections we provide details on how to launch VOSySmonitor and with what payload it can currently run.

3.2 Hardware & Software Specification of Tested Platforms

Since the Rhea platform is not available and the low-level firmware needs a highly accurate system for testing, the alternative platform of choice is ARM Fastmodels (Neoverse V1 Reference Design).

3.2.1 Hardware Specifications

The hardware specification for development and testing is described in Table 3.

Table 3. Hardware specifications of tested platform for VOSySmonitor.

Hardware Specifications	
Platform	ARM Fastmodels

3.2.2 Software Specifications

The software specification for running VOSySmonitor are described in Table 4.

Table 4. Software specifications of tested platform for VOSySmonitor.

Software Specifications	
VOSySmonitor	Being a bare metal application, the only software components running before are BL1 and BL2 firmware (reference can be retrieved by building the ATF code [1])

3.3 How to build the technology?

In order to build VOSySmonitor for the FastModels platform, the following command needs to be executed:

```
$ make PLAT=fvp_rdv1
```

[1] https://github.com/ARM-software/arm-trusted-firmware

3.4 How to run?

To run the VOSySmonitor bundle (which contains also the payload among other components) through the Fastmodels platform, the following command can be used:

```
$ ./fvp/models/Linux64_GCC-9.3/FVP_RD_V1 --stat --application
prebuilt/vosysmonitor_bundle.axf
```

4 OpenCL Driver for oneAPI Construction Kit

4.1 Overview

Task 3.2 delivers driver support for the AERO platform to enable standards-based acceleration via OpenCL (e.g., for TornadoVM in WP4) and SYCL to enable AERO to take advantage of the large open-source oneAPI ecosystem in order to accelerate applications from Artificial Intelligence to High Performance Computing. This OpenCL driver is built from the open-source oneAPI Construction Kit[2] (OCK), which is released under Apache 2.0 License with LLVM extensions.

AERO enables and optimizes AArch64 and RISC-V support in OCK while taking advantage of the latest LLVM updates. The work so far was mainly focussed on OpenCL CPU support (i.e. using the CPU as the main OpenCL device) to enable rapid bring-up on various AERO-like AArch64 systems (Cavium, Graviton) some of which may not have GPUs or other accelerators, and to mitigate hardware delays. The AERO work also led to some improvements added via pull requests to the Khronos OpenCL CTS GitHub[3] and the SYCL-CTS[4], both of which use Apache License 2.0.

4.2 Hardware & Software Specification of Tested Platforms

Since the Rhea platform is not available yet and to ensure portability of our work to other AArch64 systems, alternative AArch64 platforms have been used.

4.2.1 Hardware Specifications

The hardware characteristics of the tested platforms are described in the following tables.

Table 5. Hardware specifications of tested platform for OCK (Cavium).

Hardware Specifications	
System	Cavium (Vendor)
CPU	ThunderX 88XX, Model 1, 48 cores per cluster, 64-bit
Memory	32GB RAM

Table 6. Hardware specifications of tested platform for OCK (Graviton perf).

Hardware Specifications	
System	Graviton 3 (performance instance)
CPU	Vendor: ARM, AArch64, 64 Neoverse V1 cores per socket, 64-bit/32-bit, with ARMv8.4-A ISA including 4x128 bit Neon, 2×256 bit SVE, LSE, rng, bf16, int8, crypto
Memory	128GB RAM

[2] https://github.com/codeplaysoftware/oneapi-construction-kit
[3] https://github.com/KhronosGroup/OpenCL-CTS
[4] https://github.com/KhronosGroup/SYCL-CTS

Table 7. Hardware specifications of tested platform for OCK (Graviton dev).

Hardware Specifications	
System	Graviton 3 (developer instance)
CPU	Vendor: ARM, AArch64, 64 Neoverse V1 cores per socket, 64-bit/32-bit, with ARMv8.4-A ISA including 4x128 bit Neon, 2×256 bit SVE, LSE, rng, bf16, int8, crypto
Memory	8GB RAM

4.2.2 Software Specifications

Software dependencies, libraries, build tools, etc. are presented in Table 8. The requirements regarding toolchains (CMake, make, Python, Zlib) are the same as stated in LLVM docs[5].

Table 8. Software specifications of tested platform for OCK.

Software Specifications	
LLVM	17.0.3+ and 18.1.5+ - ongoing work to support LLVM tip (currently 19.0.0)
CMake	>= 3.20.0
Python	>= 3.8
GCC	>=7.4
Gnu Binutils	>=2.17
Zlib	>=1.2.3.4

4.3 How to build the technology?

General build instructions for the oneAPI Construction Kit and all its components can be found on its GitHub landing page[6]. Note that work is ongoing on improving the build process to have just one script/cmake command to pull in all dependencies and build OCK.

Currently, the steps to build the OCK OpenCL driver are the following:

```
$ clone https://github.com/KhronosGroup/SPIRV-Tools
$ cmake  -DCMAKE_INSTALL_PREFIX=$PWD/build/install  -G Ninja  -S . -B build
$ cmake –build .
$ git clone https://github.com/llvm/llvm-project.git
$ cd llvm-project
$ cmake -S llvm -B build -G Ninja -DLLVM_ENABLE_PROJECTS="clang" \
 -DCMAKE_BUILD_TYPE=Release -DLLVM_ENABLE_ASSERTIONS=O \
 -DLLVM_ENABLE_ZLIB=Off -DLLVM_ENABLE_ZSTD=Off \
 -DCMAKE_INSTALL_PREFIX=./build/install
$ cmake --build .
$ cd ..
$ git clone https://github.com/codeplaysoftware/oneapi-construction-kit
$ cd oneapi-construction-kit
$ cmake -B build -S . -DCA_ENABLE_API=cl -GNinja \
```

[5] https://llvm.org/docs/GettingStarted.html#software

[6] https://github.com/codeplaysoftware/oneapi-construction-kit

```
 -DCA_CL_ENABLE_ICD_LOADER=ON -DOCL_EXTENSION_cl_khr_command_buffer=ON \
 -DOCL_EXTENSION_cl_khr_command_buffer_mutable_dispatch=ON \
 -DOCL_EXTENSION_cl_khr_extended_async_copies=ON  \
 -DSpirvTools_spirv-as_EXECUTABLE=$VULKAN_SDK/tools/spirv-as  \
 -DCMAKE_BUILD_TYPE=Release -DCA_ENABLE_DOCUMENTATION=Off \
 -DCMAKE_INSTALL_PREFIX=$PWD/build/install \
 -DCA_LLVM_INSTALL_DIR=$LLVMInstall
$ export OCL_ICD_FILENAMES=$PWD/build/lib/libCL.so.4.0
```

At this point, the OpenCL driver should be found and be usable by OpenCL applications.

4.4 How to run?

```
$ make tests
```

The OCK OpenCL driver should be able to be queried using tools such as clinfo[7] and sycl-ls (from DPC++).

OCK comes with its own testing framework to test its various components. The instructions[8] include how to build/run UnitCL (OCK suite) and the OpenCL CTS to test the OpenCl driver. In the context of AERO, the OpenCL driver has been tested also through a CI job running TornadoVM on OpenCL (as described in Deliverable D4.1) and through running SYCL tests from DPC++.

[7] https://github.com/Oblomov/clinfo

[8] https://github.com/codeplaysoftware/oneapi-construction-kit/blob/main/scripts/testing/README.md

5 Homomorphic Encryption – CKKS Benchmark

5.1 Overview

Security extensions concern two major aspects: (i) and (ii) Homomorphic Encryption (HE).:

- **HW-accelerated security functions**. A suite of HW-accelerated solutions for hashing (SHA2 and SHA3), symmetric (AES) and asymmetric encryption (ECC), and random number generation has been developed by UNIPI and made available for integration in Rhea during the EPI1 project, along with generic low-level drivers. In the context of AERO, these low-level drivers are being optimized to exploit the final architecture of the Rhea processor.
- **Homomorphic Encryption (HE)**. HE is a form of encryption that allows computations to be performed on encrypted data, thus preserving user privacy when processing is outsourced to cloud services. Microsoft SEAL[9] is an open-source leveled fully HE library that allows additions, subtractions, multiplications, and rotations to be performed on encrypted integers (BFV and BGV) or real numbers (CKKS). It also provides the lightweight counterpart Microsoft SEAL Embedded[10] supporting the CKKS scheme, useful for edge devices with limited resources. The library can be compiled and run in various environments, including x86_64 and AArch64. On Intel processors with AVX-512, Microsoft SEAL can also utilize the open-source Intel HE Acceleration Library (HEXL)[11], which provides efficient implementations of low-level kernels and cryptographic primitives.

  CKKSBenchmark is a complete and configurable benchmark of the CKKS primitive functions available to the user on x86_84 and AArch64 platforms, developed to investigate the library performance and identify the optimization strategies. It calculates the expression –(A2+B*C+coeff_d*D+coeff_u), followed by a back-and-forth rotation of one position of the result, using random input vectors. CKKSBenchmark is open source and available in the AERO GitHub repository space https://github.com/AERO-Project-EU/CKKSBenchmark. It is licensed under Apache 2.0.

5.2 Hardware & Software Specification of Tested Platforms

The specifications of the main testbed platform alternative to Rhea processor (ARMv64 Neoverse V2 Nvidia GraceHopper, selected due to the fact that Rhea is based on ARMv64 Neoverse Vectorized cores) used for homomorphic encryption are detailed below. It is worth noticing that also an Intel XEON processor was used to compare Microsoft SEAL performance on x86_64 and AArch64 architecture.

5.2.1 Hardware Specifications

The hardware specifications of the used platform are described in Table 9.

[9] https://github.com/microsoft/SEAL

[10] https://github.com/microsoft/SEAL-Embedded

[11] https://github.com/intel/hexl

Table 9. Hardware specifications of tested platform for HE.

Hardware Specifications	
System	ARM NVIDIA GraceHopper architecture
CPU	ARMv64 Neoverse V2 with 72 cores
Memory	574GB

5.2.2 Software Specifications

The software specification for installing Microsoft SEAL, and building and running the CKKSBenchmark are reported in Table 10.

Table 10. Software specifications of tested platform for HE.

Software Specifications	
Microsoft SEAL	v4.1.1
Intel HEXL	v1.2.5
CMake	>=3.13
Clang++ / g++	>=5.0 / >=6.0
OS	Linux

5.3 How to build the technology?

5.3.1 Microsoft SEAL

The following commands build and install (globally) the Microsoft SEAL library, which is required for building and running the CKSSBenchmark.

```
$ cmake -S . -B build
$ cmake --build build
$ cmake --install build
```

The CMAKE_INSTALL_PREFIX option allows installing Microsoft SEAL locally. For example, the following commands install the library to ~/mylibs/.

```
$ cmmake -S . -B build -DCMAKE_INSTALL_PREFIX=~/mylibs/
$ cmake --build build
$ cmake --install build
```

The use of Intel HEXL acceleration library can be enabled at building time with the option SEAL_USE_INTEL_HEXL as follows:

```
$ cmake -S . -B build -DSEAL_USE_INTEL_HEXL=on
```

5.3.2 CKKSBenchmark

CKKSBencmark has several configuration parameters that can be modified in the file 1_kernel_benchmark_expr.cpp. These parameters are described in Table 11. Each benchmark setting, and the related SEAL context, can be configured by adding an element to the benchmark_setting vector. The available parameters are presented in Table 12.

Table 11. CKKSBenchmark configuration parameters.

Parameter	Default	Description
BENCH_UNIT	0	Select the measurement unit: 0 for us, 1 for clock_cycles
run	10	Number of benchmark repetitions per setting
num_bin	10	Number of elements in the input vector. It must be <= poly_modulus_degree/2 for each setting
range_limit	100.0	Random input interval bounds [-range_limit, range_limit]
csv_output	false	Output files in csv format, one per setting
benchmark_settings	*[empty]*	Vector of structs that define the benchmarking settings (see Table 12)

Table 12. benchmark_setting configuration parameters.

Parameter	Values	Description
encryption_mode	symmetric / asymmetric	Use of symmetric key or public/private keys
sec_level	seal::sec_level_type::tc128 / tc192 / tc256	Security level in bits
poly_modulus_degree	4096 / 8192/ 16384 / 32768	Degree of power-of-two cyclotomic polynomial used in SEAL context
modulus_bit_sizes	e.g. {60, 40, 40, 60}	Bit-lengths of prime numbers multiplied for obtaining the coeff_modulus parameter. Max size of each factor is 60 bits; sum of bit-lengths limited for each poly_modulus_degree and security level (e.g. for 8192, tc::128 max-sum219)

The following commands build the benchmark. The option SEAL_ROOT is required only if a local installation of Mircrosoft SEAL library is used.

```
$ cmake -S . -B build -DSEAL_ROOT=<path/to/local/install>
$ cmake --build build
```

5.4 How to run?

To run the benchmark, a user can run the following command in the root directory of CKKSBenchmark.

```
$ build/CKKSBenchmark
```

For each setting the benchmark outputs the Avg, S_dev, Max, and Min execution time according to the configured unit. It also provides the Run value to indicate the number of iterations run without errors. Additionally, Avg_o, Outlier, and Outlier % are provided, representing respectively the average execution time after outliers removal, the number and the percentage of outliers extracted using the interquartile approach.

When configured, a csv file is created for each benchmarked setting in the output folder.

6 RISC-V Virtualization and Emulation

6.1 Overview

RISC-V QEMU in Kubernetes[12] provides an easy method for deploying RISC-V QEMU[13] instances in Kubernetes. Using Helm[14], users can quickly configure and deploy a RISC-V VM and access its console through a web interface. The Helm chart is compatible with Knot[15]. Pre-compiled configurations support RVV 1.0 and RVV 0.7.1.

AERO partners and technology users can use the QEMU instances to emulate a development and testing platform in existing clouds. This technology is publicly available at https://github.com/CARV-ICS-FORTH/qemu-riscv64),

6.2 Hardware & Software Specification of Tested Platforms

The emulator uses QEMU on x86 platforms. We have tested the emulator with instances running on the FORTH x86 cluster, and amazon cloud.

6.3 How to build the technology?

When running Kubernetes in Docker Desktop[16], users can run the following and visit http://localhost:8080:

```
$ helm install myvm ./chart/qemu-riscv64
```

To run a second instance, users specify a different port:

```
% helm install myvm ./chart/qemu-riscv64 --set service.port=8081
```

If running Kubernetes with minikube[17], the above commands apply, but users also need to enable service forwarding to localhost with minikube tunnel.

Details on configuration flags, links to bootable data images, and more information is provided as a user guide in the repository

6.4 How to run?

Two basic containers for each VM are used:

- The launcher contains the QEMU binary. A script starts QEMU automatically when it runs, using environmental variables to define the number of CPUs, the memory, and other parameters. QEMU is configured to serve its console via telnet at local port 10023. Networking is handled by

[12] https://kubernetes.io/
[13] https://www.qemu.org/
[14] https://helm.sh/
[15] https://github.com/CARV-ICS-FORTH/knot
[16] https://www.docker.com/products/docker-desktop/
[17] https://minikube.sigs.k8s.io/

passt[18], so the VM gets the same IP as the pod and can communicate with other VMs in the same Kubernetes environment.

- The console is based on gotty[19]. Gotty runs a local command (in this case telnet), captures input and output streams, exposes them via websockets, and then offers a web-based terminal emulator[20] for the user to interact with what has been run.

The root filesystem used by QEMU is also packaged up in a container image (which we call data), using a similar method to KubeVirt[21]. Before each VM starts, the data container copies its contents to an ephemeral volume that is shared with the launcher. As data contains only VM-specific files (it can optionally also hold a custom kernel and BIOS), we copy over a BusyBox[22] binary before it starts, so it can run cp to populate the shared volume.

These images and VM can be launched using Knot.

[18] https://passt.top/passt/about/
[19] https://github.com/sorenisanerd/gotty
[20] https://github.com/xtermjs/xterm.js
[21] https://github.com/kubevirt/kubevirt
[22] https://busybox.net/

7 Summary

This deliverable presents results and modules developed during the reporting period (M18) within WP3: Operating Systems, Firmware, Drivers and Security. Some of the planned work was necessarily postponed due to the delay in the availability of the Rhea platform. However, most of the components planned were successfully designed and developed using alternative platforms. The components described are available in the AERO project repositories, and this document also includes pointers and instructions on how to build and test each component.