The Omnivisor: A real-time static partitioning hypervisor extension for heterogeneous core virtualization over MPSoCs (Artifact)

Daniele Ottaviano
Università degli Studi di Napoli Federico II, Italy

Francesco Ciraolo
Boston University, MA, USA

Renato Mancuso
Boston University, MA, USA

Marcello Cinque
Università degli Studi di Napoli Federico II, Italy

Abstract

Following the needs of industrial applications, virtualization has emerged as one of the most effective approaches for the consolidation of mixed-criticality systems while meeting tight constraints in terms of space, weight, power, and cost (SWaP-C). In embedded platforms with homogeneous processors, a wealth of works have proposed designs and techniques to enforce spatio-temporal isolation by leveraging well-understood virtualization support. Unfortunately, achieving the same goal on heterogeneous MultiProcessor Systems-on-Chip (MPSoCs) has been largely overlooked. Modern hypervisors are designed to operate exclusively on main cores, with little or no consideration given to other co-processors within the system, such as small microcontroller-level CPUs or soft-cores deployed on programmable logic (FPGA). Typically, hypervisors consider co-processors as I/O devices allocated to virtual machines that run on primary cores, yielding full control and responsibility over them. Nevertheless, inadequate management of these resources can lead to spatio-temporal isolation issues within the system. In this paper, we propose the Omnivisor model as a paradigm for the holistic management of heterogeneous platforms. The model generalizes the features of real-time static partitioning hypervisors to enable the execution of virtual machines on processors with different Instruction Set Architectures (ISAs) within the same MPSoC. Moreover, the Omnivisor ensures temporal and spatial isolation between virtual machines by integrating and leveraging a variety of hardware and software protection mechanisms. The presented approach not only expands the scope of virtualization in MPSoCs but also enhances the overall system reliability and real-time performance for mixed-criticality applications. A full open-source reference implementation of the Omnivisor based on the Jailhouse hypervisor is provided, targeting ARM real-time processing units and RISC-V soft-cores on FPGA. Experimental results on real hardware show the benefits of the solution, in terms of the seamless launch of virtual machines on different ISAs, and spatial/temporal isolation, enhanced with regulation policies.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture

Keywords and phrases Mixed-Criticality, Embedded Virtualization, Real-Time Systems, MPSoCs.

Digital Object Identifier 10.4230/DARTS.10.1.4

Funding This work is partially supported within the MICS (Made in Italy – Circular and Sustainable) Extended Partnership and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 – D.D. 1551.11-10-2022, PE00000004), and it has been carried out within the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 - EUROfusion) and by the National Science Foundation (NSF) under grant number CNS-2238476. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the EU or the European Commission or the NSF. Neither any of earlier can be held responsible for them.

Dagstuhl Artifacts Series
Schloss Dagstuhl – Leibniz-Zentrum für Informatik,
Dagstuhl Publishing, Germany

Related Article Daniele Ottaviano, Francesco Ciraolo, Renato Mancuso, and Marcello Cinque, "The Omnivisor: A Real-Time Static Partitioning Hypervisor Extension for Heterogeneous Core Virtualization over MPSoCs", in 36th Euromicro Conference on Real-Time Systems (ECRTS 2024), LIPIcs, Vol. 298, pp. 7:1–7:27, 2024. https://doi.org/10.4230/LIPIcs.ECRTS.2024.7

Related Conference 36th Euromicro Conference on Real-Time Systems (ECRTS 2024), July 9–12, 2024, Lille, France

1 Scope

The Omnivisor is a model that extends the partitioning hypervisor to enable the utilization of heterogeneous cores over MultiProcessor Systems-on-Chip. Specifically, it provides the ability to transparently start VMs on asymmetric cores (e.g. ARM64, ARM32, and RISC-V) while assuring temporal and spatial isolation between VMs.
The artifact associated with our paper aims to provide transparency and reproducibility to our research findings, specifically focusing on three experiments presented in our paper:

1. **Boot Times:** Measure the boot times of VMs on different cores.
2. **Isolation Experiments:** Evaluate isolation properties of Omnivisor.
3. **Taclebench Experiments:** Benchmarking using Taclebench on different cores.

Boot Times, Isolation Experiments, and Taclebench Experiments are presented in Figures 4, 6, and 7 of our paper.

1.1 Detailed Experiments Description

1. **Boot Times.** The experiments measure the time needed to run a simple bare-metal application of different sizes as a VM on different processing cores, specifically on Cotex-a53(APU), Cortex-R5F(RPU), and Pico32 (RISC-V soft-core). To do it, first, we compile 10 images of different sizes (1MB, 10MB, 20MB, 30MB, 40MB, 50MB, 60MB, 70MB, 80MB, 90MB) for each processor (APU, RPU, RISC-V) using the compiling tools we have integrated into the Omnivisor repository. Then, we use the Jailhouse command line interface (create, load, and start) and the Omnivisor extension functionalities to launch these VM Images on the cores, and we leverage the global timers in the platform to capture the boot times. Each VM Image is launched 100 times in order to have a statistically significant number of experiments.
2. **Isolation Experiments.** The experiments measure the execution time of a simple periodic task implemented in a VM that runs on remote cores (RPU and RISC-V). We consider various scenarios where, while the VM under test is running, a disturbance code is executed on different cores (APU, RPU1, FPGA). To do so, first, we compile the VM under test for RPU and RISC-V using the compiling tools we have integrated in the Omnivisor repository. Then, we start the VM under test on one of the cores (RPU, RISC-V) and after a few seconds we start the disturbance application on one of the other cores (APU, RPU1, FPGA). After repeating the experiments for each combination of core under test and disturbance core, we recompile the Omnivisor adding the spatial isolation features (XMPUs) and we repeat the tests. Finally, we enabled the Omnivisor with both temporal (QoS) and spatial (XMPUs) isolation features and repeated the tests again.
3. **Taclebench Experiments.** The experiments measure the execution time of the entire Taclebench suite executed on both the remote cores (RPU and RISC-V) while changing the QoS configuration using the Omnivisor interface to reach controlled degradation. We first compile the Taclebench suite for both the cores using the compiling tools integrated in the

Omnivisor repository. For the RISC-V core we removed the test that requires a floating point extension since the PICO32 RISC-V core used doesn't have it. After that, we launch the entire suite on the cores under test (RPU, RISC-V) both in isolation and when the other cores (APU, RPU1, FPGA) cause interference running membomb applications. We repeated the tests 30 times to produce statistically significant results. Then we run a binary search algorithm that modifies the memory bandwidth assigned to the disturbance cores to reach a controlled slowdown on the application of the 20

1.2 Experimental Findings

1. **Boot Times**:
   - We demonstrate that booting a VM on a remote core using Omnivisor is comparable in time to booting a VM via Jailhouse on a main core.
2. **Isolation Experiments**:
   - We first demonstrate that without the Omnivisor protection mechanism enabled, the disturbance cores are able to crash the VM under test in most cases.
   - Then, we demonstrate that if only spatial isolation is provided, even if the VM doesn't crash anymore, the disturbance cores are able to delay the execution time of the VM under test.
   - Finally, we demonstrate that using the Omnivisor complete implementation of spatial and temporal isolation, the VM under test presents only a negligible delay in execution time.
3. **Taclebench Experiments**:
   - The objective of these experiments is twofold: first, to demonstrate how the Omnivisor can induce controlled degradation in the execution time of a VM running on remote cores, and second, to elucidate how the Omnivisor streamlines the parameter tuning process for achieving an acceptable performance degradation level.

2 Content

2.1 Repositories

The Repositories used for this artifact are the following:

- Omnivisor [4]: The repository containing the building system for Omnivisor. The purpose of this repository is to automate the building of a working environment to use/test Omnivisor.
- Jailhouse-Omnivisor [5]: The repository containing the features included in the Jailhouse hypervisor to manage remote cores using the Omnivisor model.
- Test Omnivisor Host [3]: The repository containing the scripts that run on the Host PC linked to the board under test. It contains all the scripts to start the experiments and visualize the results
- Test Omnivisor Guest [2]: The repository containing the scripts that run directly on board (guest).

2.2 Software Artifacts

The following software artifacts are produced for the board:

- **Linux Image**: Linux v5.15, compiled with `jailhouse_kria_buildroot_defconfig` configuration file you can find in the following directory: [4] `Omnivisor/environment/kria/jailhouse/custom_build/linux/arch/arm64/configs/`. Essential configurations required for Jailhouse compatibility include `CONFIG_OF_OVERLAY`, `CONFIG_KALLSYMS_ALL`, and `CONFIG_KPROBES`, while the rest remain as per the default configuration provided by Xilinx for its board.

- **BOOT.BIN**: The BOOT.BIN file is the binary file used to boot the board. It is composed by several files, most of them have been retrieved from the Board Support Package (BSP) of the used platform, and are used unmodified, while others are procued ad-hoc for Omnivisor:
  - ARM-Trused-Firmware (`bl31.elf`): The Arm Trusted Firmware have been slightly modify to give the Omnivisor the capability to access power management of the cores. The used version can be found in [1].
  - The PMU Firmware (`pmufw.elf`): The PMU firmware is unmodified. It is taken from the BSP of the board.
  - The ZynqMP Firmware (`zynqmp_fsbl.elf`): The ZynqMP firmware is unmodified. It is taken from the BSP of the board.
  - Device Tree (`system.dtb`): The device tree has been slightly modified as required by the Jailhouse Hypervisor to reserve a section of the memory for the Hypervisor (see reserved-memory field) in: [4] `Omnivisor/environment/kria/jailhouse/output/boot/sources/system.dts`.
  - FPGA bitstream (`system.bit`): The bitstream used in the test comprehends a Pico32 and three AXI Traffic generators.
- **Root Filesystem**: The filesystem is generated using the Buildroot project, with the configuration file `jailhouse_kria_buildroot_defconfig` in: [4] `Omnivisor/environment/kria/jailhouse/custom_build/buildroot/configs/`
  - The jailhouse_Omnivisor [5] is included under the `/root` directory.
  - The test_omnivisor_guest repository [2] is included under `/root/tests`. The latter contains the scripts enabling the host to orchestrate the experiments.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS).

3.1 Environment Setup

All the software needed to launch the experiments is included in the provided artifact link. Once downloaded, set up the environment by following these steps:

1. Uncompress the artifact files on your host machine.
2. Copy all the files in the `SD_Board` directory to an SD card and insert the card into the board (Sec. 4).
3. Connect the board to the host machine using a USB cable, and to the network using a LAN cable. Power on the board.
4. Once the board is running, obtain its IP address and insert it into the `kria-jailhouse.sh` configuration file (configure a static IP to avoid repeating this step). This file is located in the following directory on the host machine:
   `omnivisor_artifact/environment/kria/jailhouse/environment_cfgs/`.
5. Configure the board-specific information in the `board_info.sh` file, located in the host machine directory:
   `omnivisor_artifact/tests/test_omnivisor_host/utility`
6. Update the directory information (check the `OMNIVISOR_DIR` variable) in the `default_directories.sh` file, located in the following host machine directory:
   `omnivisor_artifact/tests/test_omnivisor_host/utility`
7. Navigate to the test directory:
   `cd /home/reviewer/omnivisor_artifact/tests/test_omnivisor_host`
8. To recreate the plots with new data, activate the Python virtual environment `.venv` by typing:
   `source .venv/bin/activate`

3.2 Building from Scratch

Alternatively, starting from the Omnivisor repository [4], all the necessary artifacts can be produced and tested on a different server+board environment from scratch. Refer to the `README.md` file in the repository [4] for detailed instructions on generating all the files discussed in Sec. 2.2 for the supported boards.

3.3 Usage Instructions

Once the environment is set up, for detailed information on starting the experiments read the `README.md` file at [3]. The documentation also provides results claims, and a step-by-step guide to reproduce the experiments.
If you downloaded the artifact the images used for the tests are already integrated in the board's file system and the fastest way to reproduce the experiments is by using these images. For your convenience, we provide one script for each experiment, which replicates exactly the steps needed to achieve the results shown in the paper:

1. `test_omnnivisor_host/experiments/boot_exp/ecrts_boot_tests.sh`
2. `test_omnnivisor_host/experiments/isolation_exp/ecrts_isolation_tests.sh`
3. `test_omnnivisor_host/experiments/taclebench_exp/ecrts_taclebench_tests.sh [-p] [-b]`

Experiment	Args	Expected time
Boot		40 minutes
Isolation		14 minutes
Taclebench		90 minutes
Taclebench	-p	$>$ 40 hours
Taclebench	-f	$>$ 3 days

To accommodate the time-intensive nature of the complete Taclebench experiment, we offer three versions for evaluation. The first version prioritizes speed by bypassing binary search, utilizing pre-determined memory bandwidth allocation values as depicted in Figure 7 of our paper, and conducting single-run executions without repetitions. (time $=$ 35min):

- `test_omnnivisor_host/experiments/taclebench_exp/ecrts_taclebench_tests.sh`

The second version mirrors the first but incorporates 30 repetitions for each benchmark to enhance statistical robustness. For running each benchmark (on both RPU and RISCV) with 30 repetitions (time $=$ 18 hours):

- `test_omnnivisor_host/experiments/taclebench_exp/ecrts_taclebench_tests.sh -p`

Lastly, the third version encompasses the binary search component to optimize memory bandwidth allocation. This version is the most comprehensive but also the most time-consuming, requiring approximately 4 days for completion. For running each benchmark (on both RPU and RISCV) with 30 repetitions and integrating the binary search, execute: (time $=$ 4 days)

- `test_omnnivisor_host/experiments/taclebench_exp/ecrts_taclebench_tests.sh -b`

The raw results are located in the `./result` directory and can be visualized using the notebook `./notebooks/Omnivisor_test_plots.ipynb`. Alternatively, you can generate the images using the Python scripts located in the `./notebooks/ directory`, specifically `plot_boot_exp.py`, `plot_isolation_exp.py`, and `plot_taclebench_exp.py`. The resulting images will be stored in `./notebooks/imgs`.

If instead, you want to re-compile the images from scratch, load them on the board, and replicate the test step-by-step, follow these instructions:

3.4 Step-by-Step Instructions

3.4.1 Boot Times (Figure 4)

1. **Setup:** To re-compile the images used in the boot experiments, move to the following directory: `./experiments/boot_exp/`. Here you can find a `README.md` explaining the procedure to compile the images.
2. **Execution:** To calculate the boot times you can use the script `start_boot_exp.sh`. The `README.md` explains in detail how to use it.
3. **Analysis:** After running the experiments, compare the boot times obtained with those reported in Figure 4 of our paper. The images can be visualized in the notebook: `./notebooks/Omnivisor_test_plots.ipynb` or via the python script: `./notebooks/plot_boot_exp.py`.

3.4.2 Isolation Experiments (Figure 6)

1. **Setup:** To re-compile the images used in the isolation experiments, move to the following directory: `./experiments/isolation_exp/`. Here you can find a `README.md` explaining the procedure to compile the images.
2. **Execution:** Run the provided scripts or commands in the `README.md` to conduct the Isolation Experiments.
3. **Analysis:** After completing the experiments, analyze the isolation metrics obtained and compare them with the results presented in Figure 6 of our paper. The images can be visualized in the notebook `./notebooks/Omnivisor_test_plots.ipynb` or via the python script `./notebooks/plot_isolation_exp.py`.

3.4.3 Taclebench Experiments (Figure 7)

1. **Setup:** To re-compile the images for all the benchmarks used in the Taclebench experiments, move to the following directory: `./experiments/taclebench_exp/`. Here you can find a `README.md` explaining the procedure to compile the images.
2. **Execution:** Run the provided scripts or commands in the `README.md` to conduct the Taclebench Experiments. Given that the experiment may require a significant amount of time to complete, we offer a version of the script that bypasses the binary search process to determine the optimal configuration. Instead, it directly utilizes the configuration identified in our paper, streamlining the replication process.
3. **Analysis:** Analyze the performance metrics obtained from the Taclebench Experiments and compare them with the results reported in Figure 7 of our paper. The images can be visualized in the notebook `./notebooks/Omnivisor_test_plots.ipynb` or via the python script `./notebooks/plot_taclebench_exp.py`.

4 Tested platforms

The board tested is the Xilinx Kria™ KV260 equipped with:

- quad-core ARM Cortex-A53 (APUs)
- dual-core ARM Cortex-R5F (RPUs)
- 16nm FinFET + Programmable Logic (FPGA)

5 License

All the repositories of the artifacts are licensed under the GNU General Public License v3.0 and v2

6 MD5 sum of the artifact

1dfddc064f61577892e471bc2f15f835

7 Size of the artifact

12.52 GiB

References

1 Daniele Ottaviano. ARM Trusted Firmware patched for Omnivisor. Accessed: May 27, 2024. URL: https://github.com/DanieleOttaviano/arm-trusted-firmware.

2 Daniele Ottaviano. Test Omnivisor Guest. Accessed: May 7, 2024. URL: https://github.com/DanieleOttaviano/test_omnivisor_guest.

3 Daniele Ottaviano. Test Omnivisor Host. Accessed: May 7, 2024. URL: https://github.com/DanieleOttaviano/test_omnivisor_host.

4 Daniele Ottaviano. The Omnivisor (building system). Accessed: May 27, 2024. URL: https://github.com/DanieleOttaviano/Omnivisor.

5 Daniele Ottaviano. The Omnivisor (Jailhouse extension). Accessed: May 7, 2024. URL: https://github.com/DanieleOttaviano/jailhouse.