Towards EXtreme scale Technologies and Accelerators for euROhpc hw/Sw Supercomputing Applications for exascale

WP6 Applications and use cases

D6.3 Final Assessment and Guidelines

This project has received funding from the European Union's Horizon 2020 research and innovation programme, EuroHPC JU, grant agreement No

TEXTAROSSA

Towards EXtreme scale Technologies and Accelerators for euROhpc hw/Sw Supercomputing Applications for exascale

Grant Agreement No.: 956831

Deliverable: D6.3 Final Assessment and Guidelines

Project Start Date: 01/04/2021 **Duration**: 36 months

Coordinator: *AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE, L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE - ENEA, Italy.*

Deliverable No	D6.3
WP No:	WP6
WP Leader:	PSNC
Due date:	M36 (March 31, 2024)
Delivery date:	31/03/2024

Dissemination Level:

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

This project has received funding from the European Union's Horizon 2020 research and innovation programme, EuroHPC JU, grant agreement No

DOCUMENT SUMMARY INFORMATION

Project title:	Towards EXtreme scale Technologies and Accelerators for euROhpc hw/Sw Supercomputing Applications for exascale
Short project name:	TEXTAROSSA
Project No:	956831
Call Identifier:	H2020-JTI-EuroHPC-2019-1
Unit:	EuroHPC
Type of Action:	EuroHPC - Research and Innovation Action (RIA)
Start date of the project:	01/04/2021
Duration of the project:	36 months
Project website:	textarossa.eu

WP6 Applications and use cases

Deliverable number:	D6.3					
Deliverable title:	Final Assessment and Guidelines					
Due date:	M36					
Actual submission date:	M36					
Editor:	Pasqua D'Ambra					
Authors:	Massimo Bernaschi, Mauro Carrozzo, Alessandro Celestini, Pasqua D'Ambra, Daniel Jaschke, Paolo Palazzari, Michal Kulczewski, Ariel Oleksiak, Miłosz Ciżnicki, Wojciech Szeliga, Daniel Jaschke, Alessandro Lonardo, Luca Pontisso, Cristian Rossi, Francesco Simula, Martin Kuehn					
Work package:	WP6					
Dissemination Level:	Public					
No. pages:	136					
Authorized (date):	30/03/2023					
Responsible person:	Pasqua D'Ambra					
Status:	Plan	Draft	Working	Final	[✓ Submitted]	Approved

Revision history:

Version	Date	Author	Comment
0.1	2023-10-27	Pasqua D'Ambra	Draft structure
0.2	2024-02-15	Pasqua D'Ambra	CNR section
0.3	2024-03-04	Sergio Saponara	UNIPI (CINI)
0.4	2024-03-15	Carlos Álvarez	RAIDER OmpSs@FPGA results
0.5	2024-03-15	Antonio Filgueras	OmpSs@FPGA IDV-E results
0.6	2024-03-15	All	Remaining contributions
0.7	2024-03-18	Michał Kulczewski	Ready for first internal review

0.8	2024-03-28	All	Final remaining contributions, addressing internal reviewers' comments
0.9	2024-03-29	Pasqua D'Ambra	Ready for submission

Quality Control:

Checking process	Who	Date
Checked by internal reviewer 1		25/03/2024
Checked by internal reviewer 2		25/03/2024
Checked by Task Leader	Pasqua D'Ambra	29/03/2024
Checked by WP Leader	Michał Kulczewski	29/03/2024
Checked by Project Coordinator	Massimo Celino	30/03/2024

COPYRIGHT

ACKNOWLEDGEMENTS

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement no 956831. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Italy, Germany, France, Spain, Poland.
Please see https://TEXTAROSSA.eu for more information on the TEXTAROSSA project.
The partners in the project are AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE, L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE (ENEA), FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V. (FHG), CONSORZIO INTERUNIVERSITARIO NAZIONALE PER L'INFORMATICA (CINI), INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE (INRIA), BULL SAS (BULL), E4 COMPUTER ENGINEERING SPA (E4), BARCELONA SUPERCOMPUTING CENTER-CENTRO NACIONAL DE SUPERCOMPUTACION (BSC), INSTYTUT CHEMII BIOORGANICZNEJ POLSKIEJ AKADEMII NAUK (PSNC), ISTITUTO NAZIONALE DI FISICA NUCLEARE (INFN), CONSIGLIO NAZIONALE DELLE RICERCHE (CNR), IN QUATTRO SRL (in4). Linked third parties of CINI are POLITECNICO DI MILANO (CINI-POLIMI), Università di Torino (CINI-UNITO) and Università di Pisa (CINI-UNIPI); linked third party of INRIA is Université de Bordeaux; in-kind third party of ENEA is Consorzio CINECA (CINECA); in-kind third party of BSC is Universitat Politècnica de Catalunya (UPC).
The content of this document is the result of extensive discussions within the TEXTAROSSA © Consortium as a whole.

DISCLAIMER

Table of contents

List of Figures

List of Tables

Executive Summary

This deliverable summarizes possible adaptation of applications and/or TEXTAROSSA toolchain features which was needed to obtain benefits in terms of performance and energy efficiency, as evaluated by the KPIs defined for each use case in the previous D6.1 deliverable. Therefore, as follow up of all the WP6 activities on the applications, it provides guidelines, possible recommendations, and final evaluation of the project efforts. All use cases applied common methodology for performance and energy measurements, discussed in detail in D1.4. The tests were conducted among others on TEXTAROSSA available testbeds, including IDV-A (node with NVIDIA GPU accelerators from Atos) and IDV-E (node with FPGA accelerators from E4). The applications represent wide range of application types (HPC, AI/HDPA), scientific domains, approaches to parallelization (task-based, streaming-based, distributed/shared memory). In particular, the use cases benefit from developments around one of the three main pillars defined: heterogenous resources, mixed precision and dynamic runtime systems.
The work carried out correspond directly to the following overall project objectives:

- Energy efficiency, by the application developments;
- Sustained application performance, by the application developments;
- Seamless integration of reconfigurable accelerators, by using the APEIRON framework;
- Development of new IPs, by using INFN intra/inter-FPGA communication IP behind the APEIRON framework;
- Integrated Development Platform, by using existing IDV-A and IDV-E;
- Opening of new usage domain, by the application developments.

CINI-UNIPI provides example results of HPC services for smart cities. Particularly, the proposed application refers to processing of video-surveillance applications for smart cities, where data acquired by telecameras are first processed with YOLO (You Only Look Once) to detect scenes containing people, and then people are automatically counted, and their position tracked using DeepSORT. The latter is an evolution of the tracking algorithm SORT: Simple Online and Realtime Tracking. Profiling of the algorithm on several platforms (those with Arm N1 processors are representative of the IDV-E) allows assessing the usefulness of different technologies. Results on RISC-V based platforms are also reported.

CNR has developed computational kernels for a TEXTAROSSA MathLib specifically required in sparse matrix computations and iterative linear solvers, which are widely exploited in Scientific Computing and Data Analysis. The library was thought both as main component of the TEXTAROSSA platform and as benchmark tool for IDV-A. Focus was on GPU-kernels efficiency and on scalability when multiple GPUs, also on different computing nodes, are needed for computations because dimensions largely exceed the memory resources of a single GPU. Therefore, on IDV-A, the library stresses the GPU operation capabilities and memory/communication channels bandwidth at the node level. Performance analysis and GPU energy consumption KPIs on IDV-A are reported in section 2.2. In the same section are discussed scalability results and comparisons with the state-of-the-art Nvidia library on the Leonardo Italian supercomputer. Results demonstrate benefits of the new algorithms, specifically thought for GPU-accelerated architectures, and of the implementation design patterns, which efficiently exploit high throughput of GPUs and reduce/hide data communication among memories and computing nodes, with respect to Nvidia library implementing same computations. Power and energy consumption results of the kernels on

IDV-A are consistent with the strong and weak scalability performance and are discussed before and after the installation of the new 2-phase cooling system on the IDV-A platform.

PSNC provided UrbanAir, a HPC application for the purpose of benchmarking and hardware characterization, IDV-A in particular. The scope of UrbanAir is to predict air quality in urban environment, by detailed modelling of the wind flow over street canyons and complex building structure. In TEXTAROSSA, work was focused on GPU version, UrbanAir-gcrk, to run the kernels exploiting GPUs accelerators. The primary purpose was to increase energy efficiency and computation performance (lower time-to-solution). The secondary purpose was to exploit multi-GPUs available on multiple nodes to be able to run over domains which dimensions exceed the memory constraints of a single GPU. Results demonstrate that by adapting codes to accelerators it is possible to achieve primary and secondary goals.

FHG has evaluated the usability of reduced precision Floating Point format for seismic Reverse Time Migration calculations (RTM). A small reference implementation has been created to calculate the Marmousi reference example based on 16 bit Posit and 16 bit Float format. The quality of the results has been evaluated based on the images calculated and the numerical stability of the total energy of the source wavefields over time. Storing the wavefields in Posit 16 bit delivered acceptable images. Performing the calculations in Posit 16 bit too required some optimizations but finally delivered also acceptable results. Calculating the image in Posit 16 bit too revealed higher deviations from the reference result but the image quality was still acceptable. In Float 16 bit format both storage of the wavefields and the calculation of the kernel delivered acceptable images. Calculating the image in Float 16 bit too revealed higher deviations from the reference result but the image quality was acceptable.
BSC has used NBody application (as reported in section 2.12) in order to test all its framework developments along the project like the fast task scheduler IP, the OmpSs@FPGA task-based programming model, the task+stream models integration, the FPGA power measurement tools or the multinode programming model Implicit Message Passing. Also, the power and cooling efficiency of the OmpSS@FPGA model in the IDV-E node has been evaluated. Beside the results of this tasks reported in this deliverable, also the integration of the RAIDER application stream kernel with the OmpSs@FPGA task-based model was done in coordination with INFN and reported in the corresponding section 2.6.

ENEA used the Vitis High-Level Synthesis (HLS) flow to implement an image processing library (FPGA Image Processing Library - FIPLib) on the Alveo U280 board. This to showcase the benefits in terms of both speed and power usage achievable using FPGA accelerators. The FIPLib has also been used in conjunction with the APEIRON framework, developed by INFN, to demonstrate how a design could be scaled to several FPGA accelerators to sustain a higher throughput, not achievable through a single FPGA for its limited resources.

INFN provided one High Performance Data Analytics (RAIDER) and three HPC (NEST-GPU, HEP, TNM) applications as benchmarks to drive the co-design and characterization activities of project IDVs.
The RAIDER (Real-time AI-based Data analytics on hEteRogeneous distributed systems) application, described in section 2.6, is driven by the use case of a real-time particle identification system for the CERN NA62 High Energy Physics experiment. Being entirely FPGA-based, it represents the TEXTAROSSA IDV-E reference application for its characterization in terms of processing throughput and energy efficiency KPIs. RAIDER was co-designed and co-developed with the APEIRON HLS streaming programming framework aimed at the seamless scalable integration of reconfigurable accelerators. The key technology

behind the APEIRON framework enabling its scalability feature is the INFN intra/inter-FPGA communication IP, developed according to the project objective “Development of new IPs”.
NEST-GPU is a GPU-accelerated neural network simulator engine for in-silico experiments which aims for easy reconfigurability and usage by the neurophysiology practitioner while striving for high efficiency and performance. While being self-standing production-ready code, the very significant gains in power consumption and reduced runtimes that it has demonstrated against its sibling application NEST (which is CPU-only) have motivated the current effort for its integration into the larger environment managed by the NEST Initiative and are fostering its employment in a larger number of hybrid CPU+GPU HPC platforms. Simulating a neuronal system distributed over the available GPUs, the NEST-GPU application stresses both the GPU computing and the inter-GPU communication capabilities of the IDV-A node. Furthermore, the CPU-only NEST simulator engine has been used to characterize and compare the processing and energy performance of the IDV-A and IDV-E nodes, as reported in section 2.5.
Regarding High-Energy Physics (HEP) codes on heterogeneous architectures we have selected two representative applications: Pixeltrack, a track reconstruction algorithm, and CLUE, a cluster algorithm for high-granularity calorimeters, both developed for the for the CERN CMS experiment (see section 2.4). The applications are natively heterogeneous, being implemented with the Alpaka [1] library, and have been run both in CPU and in CPU+GPU configurations in the IDV-E and IDV-A nodes. Finally, the tensor network methods (TNM) application combines multiple simulation frontends for simulation quantum systems. For TEXTAROSSA, we considered the following sub-applications: i) Quantum matcha TEA: gate-based quantum circuit emulator for digitized quantum circuits, and ii) Quantum green TEA: solver for the Schrödinger equation or Lindblad equation; within this analysis, we restrict ourselves to finding the ground state of a system. Both have been used to characterize the processing and power efficiency of hybrid CPU+GPU systems like the IDV-A, as described in section 2.7.

Introduction

Work performed in WP6 is essential to demonstrate the TEXTAROSSA outcomes in both hardware and software perspective. The applications need to use these for the final evaluation of the project, but far more important is to come up with conclusions if and how new hardware and software toolchain can produce benefits in terms of computation and energy efficiency of applications coming from different domains as well as can make available new and easy to use development and evaluation tools.

In the TEXTAROSSA project we proposed some applications related to AI (Artificial Intelligence), HPDA (High Performance Data Analytics) and HPC (High Performance Computing). Provided software represents quite a comprehensive set of different hardware used (CPU, GPU, FPGA), programming models and problems to be solved. Use cases are developed based on three distinctive approaches: i) adaptation to heterogenous resources, ii) applying posit and mixed precision, and iii) using high-level programming models and their dynamic runtime systems. Therefore, there is a different set of computational, energy efficiency and accuracy metrics defined (KPI – key performance indicator) for each of the applications, though some naturally overlaps. The KPIs were discussed in the previous D6.1 deliverable. In this document we focus on applications development, and on reporting benchmarks and results. The work carried out is related to the following project objectives:

- energy efficiency: by applications developments to adapt to heterogeneous resources, energy efficient accelerators or using mixed precision;
- sustained application performance: by applications development to adapt to more computational efficient accelerators, in some cases applying a re-design of basic algorithms in order to better exploit fine-grained parallelism of accelerators, or/and in using scheduler of task/streaming-based frameworks;
- seamless integration of reconfigurable accelerators: by using the Apeiron framework;
- development of new IPs: by testing INFN intra/inter-FPGA communication IP which works behind the APEIRON framework;
- Integrated Development Platform: by using the available IDV-A and IDV-E platforms for final benchmark results;
- opening of new usage domains: by developing applications in many different domains, e.g. climate, oil&gas, high-energy physics.

To give a global picture of the TEXTAROSSA project achievements, in Figure 1Figure we summarize all the components developed or improved in this project, as an integrated HW/SW layered architecture. At the bottom layer we have the two types of accelerated prototype heterogeneous nodes (IDV-A and IDV-E), equipped with last generation NVIDIA GPUs and Xilinx Alveo FPGAs and the two-phase cooling system developed in this project. On the top of the above infrastructure, we have a basic toolchain including standard compilers and libraries for parallel programming as well as run-time supports and tools by vendors which, where it was needed for efficiency and scalability motivations, were directly used by some higher-level software tools and applications. In the middle layer we put all the new tools proposed in this project, including structured programming environments, application-specific workflow management tools, automatic code generation tools for exploitation of mixed-precision variable storage and computations, tools and libraries for modeling and measurements of energy consumption. Finally, on the top of this infrastructure, we put the applications, which in turn can be classified as general-purpose mathematical libraries, domain-specific codes, and mini applications. All the above applications are representatives of different computational needs and were developed as significant benchmarks of some of the components of the above architecture. Their development and testing provided useful feedback for the development and setup of all the basic components of the TEXTAROSSA architecture and significant guidelines, not only for the tools developed in this project, but more generally, for efficient and scalable exploitations of current heterogeneous computing nodes in relevant application domains.

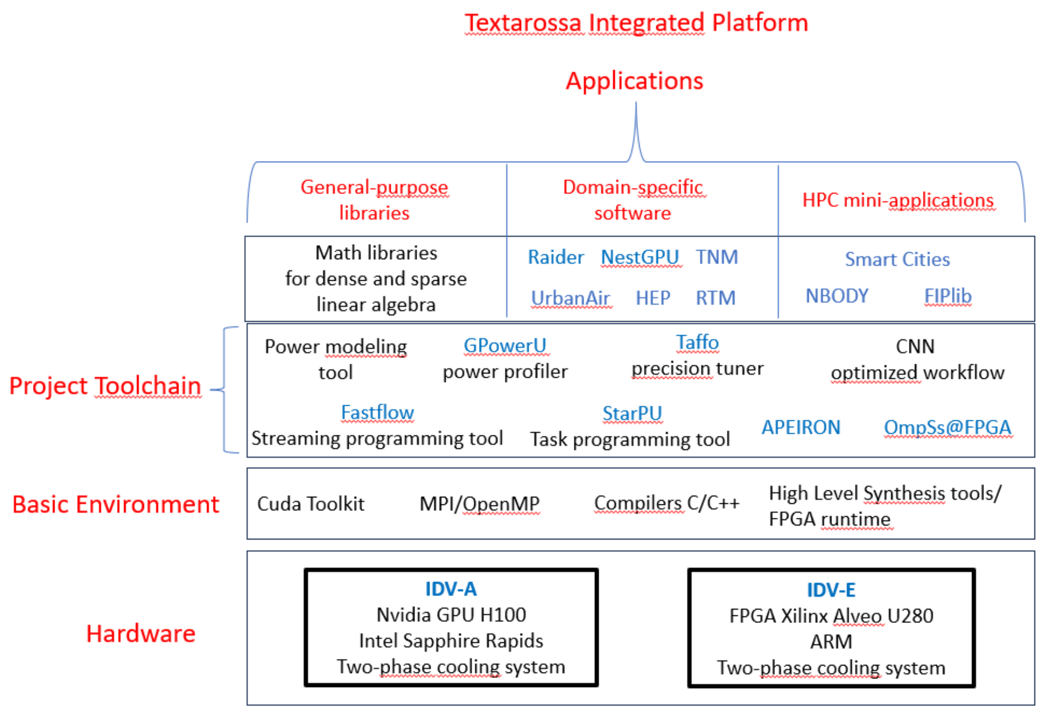


Figure 1: TEXTAROSSA Integrated Platform

This document is organized as follows. In Section 1 a briefly reminder is given on hardware being used for evaluation of applications, as well as on benchmarking methodology. In Section 2, final results of each use case are presented. Section 3 provides general summary and recommendations.

1 Methodology

1.1 Hardware

1.1.1 IDV-A

IDV-A is based on one of Atos' 1U blade servers commercialized inside the Sequana3 architecture. It consists of one Nvidia Redstone-Next GPU board equipped with four Nvidia H100 GPUs, piloted by an Atos C4E CPU board equipped with two Intel Sapphire Rapids CPUs. It is currently cooled down by a combination of water-blocks (single-phase water heat spreaders) for the CPUs and GPUs, heat pipes and a cold-plate which serves as a chassis for the boards and distributes cold water to the various heat-spreaders. More details are described in D5.1 and D1.4.

1.1.2 IDV-E

IDV-E system is delivered by E4 and it is currently available with remote access to project partners. The nodes are equipped with ARM64 and FPGAs. The choice of the system to which to apply the two-phase cooling system fell on the Ampere Mt.Collins 2U system with Ampere Altra Max processor; the main reasons are: (i) it supports a number of PCIe slots providing the possibility of adding FPGA boards (up to 3) and/or other boards if needed, (ii) it has the physical space for adding the cooling system, (iii) it presents a good match between the amount of heat to be removed and the design point of the cooling system developed in the project, (iv) it has an architecture (ARM) compatible with that of the EPI project, (v) the possibility of receiving the system in times compatible with the project (an aspect not taken for granted given the current state of shortage worldwide). As for the FPGA, the choice fell on the U280 Xilinx Passive Model, it is able to provide significant computing power and the flexibility of memory access via HBM2 or DDR protocol with a maximum consumption of 225W. This device also guarantees the use of the Vitis High Level Synthesis software stack. More details are described in D5.2 and D1.4.

1.1.3 Other architectures

To assess applications results and demonstrate the potential of node-level efficiency and scalability, when more than 1 computing node are used, some other tests have been done on some available heterogeneous computers, operated by project partners or accessed by some other grants.

- For CNR-MathLib, Leonardo is used, operated by the CINECA Italian supercomputing center and granted by an Early Access Project. The booster module used for the tests, is based on BullSequana XH2000 compute blade, it has 3456 nodes equipped with Intel Xeon Platinum 8358 32C 2.6GHz and 4 NVIDIA A100 SXM4 64GB GPUs. The system is interconnected through a Quad-rail NVIDIA HDR100 Infiniband. It is ranked 6th in the November 2023 Top 500 list.

- For Inria-MathLib, computing nodes are used from the plafrim cluster (www.plafrim.fr), which is a scientific instrument located in Bordeaux (France) and designed to support experiment-driven research in all areas of applied mathematics related to modeling and high-performance computing. We used computing nodes with NVIDIA A100 or V100 GPUs.
- For UrbanAir kernels, Altair – HPC machine operated by Poznan Supercomputing and Networking Center - is used to perform additional tests in multimode environment. The nodes are equipped with Intel Xeon Gold 6242 2.8GHz and NVIDIA V100 SXM2 GPUs.
- For n-body scalability kernel CPU implementation baseline, the Marenostrum 4 machine operated by BSC was used. Each of the computing nodes are equipped with 2 Intel Xeon Platinum 8160 CPU with 24 cores each @ 2.10GHz and 96 GB of memory. Nodes are connected using Intel OmniPath 100Gb/s network.
- For multi-node FPGA measurements, the MEEP machine, operated by BSC was used. This machine is formed by 96 compute nodes. Each one is equipped with two Intel Xeon Gold 6330 CPU @ 2.00GHz, 256 GB of main memory and 8 AMD Alveo U55c FPGA accelerator cards. FPGA accelerators are connected using 100Gb/s Ethernet.
- For the NEST-GPU, HEP and TNM (Quantum matcha TEA) applications we used the BSC DIBONA partition node with a dual-socket system with two AMD EPYC 7402 24-Core Processor and four NVIDIA A100-SXM4-40GB to collect results to use as baseline to compare against performance on the IDV-A node.
- For the TNM (Quantum green TEA) the CINECA Leonardo booster node and the Justus2 cluster node (bwHPC): 2xIntel Xeon 6252 Gold with 2x24 cores.
- For the RAIDER HPDA multi-FPGA applications, we used the INFN APE Lab small development cluster made of 4 single Intel(R) Xeon(R) Silver 4410T CPU nodes, each of them hosting one Xilinx® Alveo U200 FPGA board, with the four FPGAs interconnected in a ring topology.

1.2 Application tuning, performance, and energy measurements

In order to have meaningful and consistent benchmarking results across different use cases, a common methodology for measuring the performance and energy efficiency is proposed and discussed in detail in D1.4 and followed in D6.2. Some of the approaches are described in this document in Section 3, where each use case presents in detail the approach to obtain results and measure application-specific KPIs (proposed in D6.1). Where it was needed to follow an architecture-specific pathway and/or adaptation at the algorithmic and/or implementation level, this is discussed in detail so that this Deliverable can be read as a set of guidelines for potential users of all the HW/SW IPs developed in the project.

2 Results

2.1 Smart cities – CINI-UNIPI

Smart Cities algorithm description and implementation assumptions

CINI-UNIPI provides example results of HPC services for smart cities. The proposed application refers to processing of video-surveillance applications for smart cities, where data acquired by telecameras are first processed with YOLO (You Only Look Once) to detect scenes containing people. Then people are automatically counted and their position tracked using DeepSORT, which is an evolution of the tracking algorithm SORT: Simple Online and Realtime Tracking. YoloV5 is an algorithm that ensures low-latency in extracting the object of interest in a complex scene. YoloV5 was trained on a custom labelled dataset (to recognize specifically people and people laying down). Moreover, DeepSORT is designed to achieve real-time. Hence, the combination of both algorithms ensures a rapidity that is essential in mission-critical applications. DeepSORT takes advantage of OSNet x1.0, a convolutional neural network used for person re-identification, to identify the same person in the video stream.

Thanks to a geometrical check of the bounding boxes generated by Yolo to highlight the target of interest in the scene (people), the algorithm can also detect if people are laying down or not. This is important when monitoring zones of the city after a natural disaster (e.g. earthquake, landslide) or in a war scenario.

Profiling of the algorithm on several platforms (those with Arm N1 processors are representative of the IDV-E) allows assessing the usefulness of different technologies. Results on RISC-V based platforms are also reported.

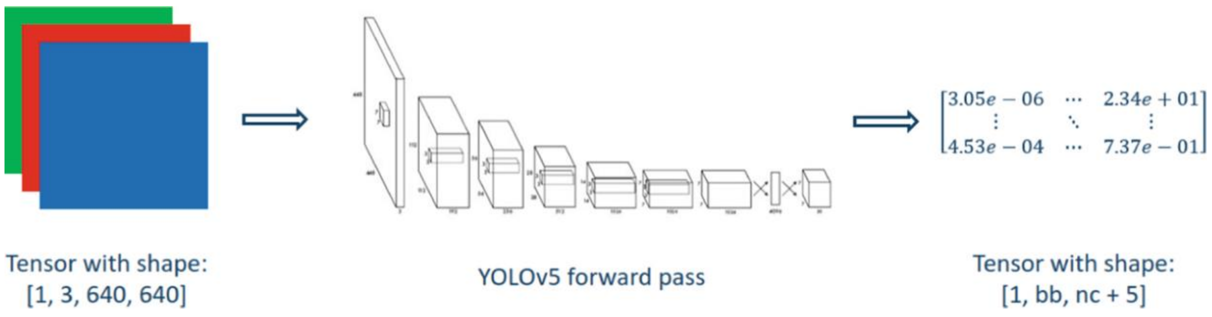
To be noted that as far as the computation partitioning between general-purpose processor and Hardware (HW) accelerator is concerned, Deep-SORT has an irregular data flow and hence is less suited for HW acceleration on FPGA than Yolo. For YoloV5 we considered both the case it is implemented SW-wise on the general-purpose processor or HW-wise via an FPGA acceleration. In this case the complete flow can be pipelined with the FPGA processing with Yolo the input video flows and then the general-purpose computing cores applying DeepSORT and other control rules.
In all cases the General Purpose CPU also implements some pre-processing of the input videos acquired from the surveillance camera.

As far as mixed-precision is concerned, the Posits were tested through the CppPosit software library developed by UNIPI [2].

Indeed, in TEXTAROSSA it is not foreseen designing a custom processor as an ASIC with dedicated HW support for new formats like Posits.

Moreover, some of the considered platforms (e.g. those with NVIDIA GPUs) consider automatically an optimization of multiple arithmetic precisions among float32, bfloat/float16 and integers (8 or lower). The native mixed-precision is more effective than emulating via software the CppPosit library on a given platform.

In the reported Figure 2 and Table 1 the results refer to the best mixed-precision configuration.



$\begin{bmatrix} 3.05e-06 & \cdots & 2.34e+01 \\ \vdots & \ddots & \vdots \\ 4.53e-04 & \cdots & 7.37e-01 \end{bmatrix}$
Tensor with shape: [1, 3, 640, 640]
YOLOv5 forward pass
Tensor with shape: [1, bb, nc + 5]

Figure 2: [SmartCities] Computation flow for the YOLOv5 step

Target platform and implementation results

We evaluated the aforementioned deep-learning application through Pytorch on several architectures. We employed architectures with and without accelerators (GPUs, FPGA), to also evaluate the performance of vector units inside the CPUs, when available. The tested CPU-only architectures are based on ARM, RISC-V and Intel.

The platforms used in the test are available at the Green Data Center of University of Pisa (GREEN DATA CENTER (unipi.it)).
Most of the reported HPC platforms, including those with ARM N1 processors, are installed and maintained at UNIPI Green Data Center by E4 and hence the results achieved are coherent with those achieved on machines made available via VPN by E4.

For ARM we have considered ARM 64 bit architectures and particularly: the ARM Neoverse N1 that is the one available in IDV-E, plus that in the Fujitsu A64FX CPU that is the ARM SVE (scalable vector extension) and finally the ARM Cortex-A72 available in Xavier NVIDIA SoC (system-on-chip) computing board.

- ARM Cortex-A72 CPU (with ARMv8.0-A 64-bit instruction set) with four cores running at 1.5 GHz, with support for the 128-bit Neon SIMD extension and 4 Gbyte of RAM.
- ARM Cortex-A78 CPU (with ARMv8.2-A 64-bit instruction set) has been also tested since is inside the AGX Orin with 12 cores.
- HPE Apollo80 system, powered by a Fujitsu A64FX, running at 2.2 GHz with support for the ARM 512-bit Scalable Vector Extension (SVE) SIMD unit.
- Ampere Altra Max with ARM Neoverse N1 CPU with support to 128-bit NEON SIMD instructions (IDV-E platform). ARM Neoverse N1 CPU being derived from Cortex-A76 supports the ARMv8.2-A 64-bit instruction set as in the Cortex-A78 mentioned above.
- HiFive Unmatched board powered by a SiFive U740 SoC running at 1.2 GHz for the scalar RISC-V unit called Arriesgado.
- Cascade Lake-based Intel Xeon Silver CPU running at 2.2 GHz with support for the 512-bit AVX SIMD instructions.

- I7-107650H processor from Intel, implemented in 14nm is a 10th generation x86 I7 family CPU with AVX2 and SSE4.2 instruction extensions (256-bit SIMD instructions), with 2.6 GHz basic frequency (boot up to 4.8 GHz).

The tested GPU-based architectures are:

- NVIDIA Jetson Orin SoC with an Ampere-based GPU
- NVIDIA Tesla T4
- NVIDIA A100

Hereafter, we report the benchmark results for the platforms listed above. In Table 1 we report the average number of frames processed per second for each platform on the whole application (preprocessing and YoloV5 + people down check + DeepSORT) application. Furthermore, we detail, for each architecture, the timing performance for the three different components of the application.

Platform	FPS	YOLO (ms)	People-down(ms)	DeepSORT (ms)
ARM Cortex-A72	0.11	7493	1.2	1107
ARM NeoverseN1	0.73	858	0.6	503
IDV-E: ARM N1+FPGA	2	(FPGA)	(FPGA)	503 (ARM)
ARM A64FX	0.43	1292	1.1	1054
RISC-V SIFIVE U7540 emulating Arriesgado#	0.006	148987	2.6	13835
AGX Orin (12 Cortex-A78 cores CPU + GPU Ampere) *	7.7	38.8	0.5	54.9
Intel i7	0.98	807	0.2	207
Intel i7 + GPU GeFore 1650Ti (Turing)*	7.3	85.4	0.3	11,9
Intel Xeon	1.88	335	0.3	197
Intel Xeon + GPU Tesla T4 *	12.8	37.8	0.3	15.1
Intel Xeon + GPU A100*	21.3	10.9	0.3	13.9

Table 1: [SmartCities] Implementation results and benchmarks for different platforms

Please note that for the 4 cases with * where we have a CPU plus a GPU accelerator, due to bottlenecks in the data transfer between the CPU part and the GPU part the overall frame-per-seconds (FPS) is lower than the inverse of the sum of the individual frame processing time (i.e. for Intel Xeon + GPU A100* the sum of frame processing time would be 25.1 ms, i.e. its inverse would be a theoretical FPS od 39.84 FPS but due to data transfer bottlenecks among CPU-GPU we measured 21.3 FPS).

Please also note that for Arriesgado scalar RISC-V emulated on SI-Five u7540 the very low FPS performance achieved when compared to other solutions is due also to the fact thet the architecture is emulated.

It is worth remembering that the Tesla A100 has fully native support of mixed precision.
A100 Tensor Core GPU performance specs [3]:

- Peak FP64 9.7 TFLOPS
- Peak FP32 19.5 TFLOPS
- Peak FP16 78 TFLOPS
- Peak BF16 39 TFLOPS
- Peak TF32 Tensor Core 156 TFLOPS
- Peak FP16 Tensor Core 312 TFLOPS
- Peak BF16 Tensor Core 312 TFLOPS
- Peak INT8 Tensor Core 624 TOPS
- Peak INT4 Tensor Core 1,248 TOPS

In Figure 2 we analyze the power efficiency for all platforms with an FPS value higher than 7, that is the minimum considered useful for a real application. The power efficiency is measured at application level, so the KPI is Frame-Per-Second/Watt or Frame/Joule being 1 Joule equal to 1 Watt * 1 second.
The results achieved show that the most efficient solution is the one with multicore Cortex-A78 64b plus NVIDIA GPU, although optimizations towards an increases power efficiency are needed since the cost is still 2 Joule for each processed Frame.
If we need to increase the performance to a real-time value for more than 20 FPS the best solution is the Intel XEON + A100 GPU with an energy cost of 2.5 Joule per processed frame.

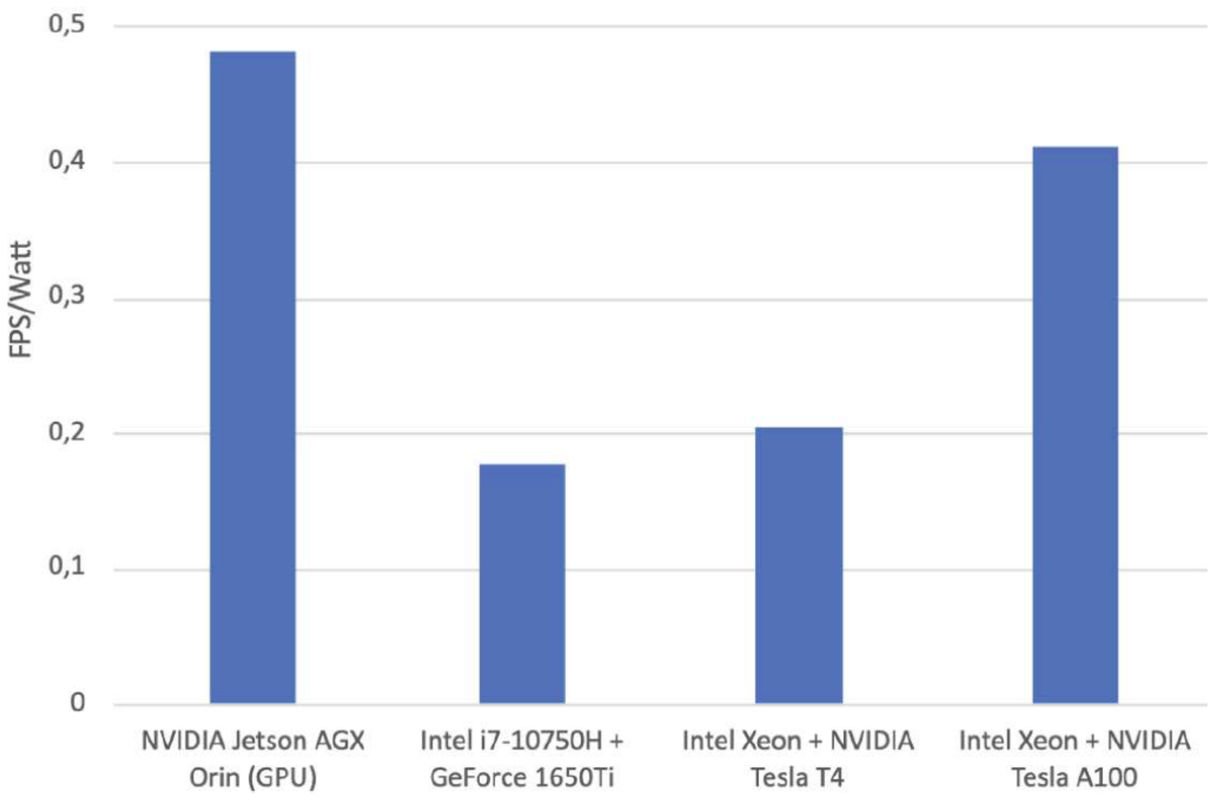


Figure 3: [SmartCities] Power efficiency measured as FPS/Watt

the higher is the better, for the platforms with FPS higher than 7 (that is the minimum considered for a practical use)

From the results achieved, the key lesson learned to set-up an edge server for surveillance applications in smart cities are:

- RISC-V computational capabilities, particularly in scalar version, and considering commercially available solutions are still far in performance from other platforms like those based on Intel and/or ARM.
- For ARM the SVE version in Fujitsu performed worse than the ARM Neoverse N1 in Ampera Altra Max (the one in IDV-E) that is not using SVE; this can be justified by the fact that the algorithms were not optimized for the scalar vector extension of the processor as implemented in Fujitsu.
- Multicore ARM Cortex-A 64b (like in the A78 with ARM v8.2 instruction set) sustained by a GPU acceleration are the most efficient solution for edge –AI/Video applications.
- Looking to edge server performance the combination of Intel Xeon plus a GPU Like Tesla A100 is the best in terms of pure performance, FPS as KPI, but is also a good solution in terms of energy efficiency, FPS/Watt KPI. Since the new ARM Neoverse V2 is appearing in the market with ARM 64b v8.4 instruction set and vectorized instruction, we expect that ARM Neoverse V2 plus GPU can be as effective as Intel Xeon plus A100 GPU.
- The availability of FPGA Xilinx Alveo in IDV-E can be exploited by porting the Yolo calculation on it while keeping the DeepSort on the ARM processor and hence a gain in speed can be achieved roughly by a factor of 2. However, the performance are still below those that can be achieved using Intel Xeon plus a GPU Like Tesla A100.
- Modern GPUs like A100 natively support mixed-precision from very low-format (from INT4 to FP64 including new FP16 or Bfloat16)
- New chip evolutions expected from main US companies will extend this mixed-precision support to microscaled Floating Point formats having also FP4 and FP8 as alternative to IEEE 754 standard, see this joint paper from Intel, Meta, Nvidia, AMD, Microsoft, Qualcomm 2310.10537.pdf (arxiv.org) entitled “Microscaling Data Formats for Deep Learning” and the open library from Microsoft GitHub - microsoft/microxcaling: PyTorch emulation library for Microscaling (MX)-compatible data formats. This demonstrates the value of an independent research on compact floating –point variants vs the IEEE 754 standard, like Posits and integration for Posits with other formats in tools like TAFFO. An idea on future work can be the combination in TAFFO of Posits16, 32 with FP32/FP64 for high accurate numerical applications and Posit8, with BFloats16, FP8, FP4, INT4, INT8 for video or AI applications where the classification accuracy can tolerate a higher numerical inaccuracy.

2.2 MathLib – CNR

In this section we discuss results obtained by CNR using the mathematical software library for hybrid architectures featuring NVIDIA GPUs at node level. As already described in previous deliverables, the CNR team has developed computational kernels required in sparse matrix computations and iterative linear solvers, which are widely exploited in Scientific Computing and Data Analysis. Our library was thought both as main component of the TEXTAROSSA platform and as benchmark tool for IDV-A. Focus was on GPU-kernels efficiency and on scalability when multiple GPUs, also on different computing nodes, are needed for computations because dimensions largely exceed the memory resources of a single GPU. Therefore, on IDV-A, the library stresses the GPU operation capabilities and memory/communication channels bandwidth at the node level. The Mathlib development toolchain includes C compilers, the Cuda Toolkit and the MPI library available as basic environment of the TEXTAROSSA platform. Our choice of the above basic tools was motivated by the need to re-use some very efficient GPU kernels already available in the library baseline (for 1 NVIDIA GPU) and to focus on algorithm and implementation scalability when very large number of nodes could be needed. Moreover, extensive use of the GPowerU project tool has been made for monitoring GPU kernels energy consumption. In the following we first discuss some adaptation and approximations needed at the algorithmic and implementation levels to have benefits from the use of heterogeneous systems and then discuss main results.

The MathLib computational kernels developed and tested in the project are the following:

- Sparse matrix – vector multiplication (SpMV);
- Sparse matrix – matrix multiplication (SpMM);
- Maximum Weight Matching in undirected graphs (MWM);
- Preconditioned Conjugate Gradient (PCG) method coupled with a matching-based Algebraic MultiGrid preconditioner. Note that in the following we use the acronym BCMG to refer to the complete solver.

We point out that the BCMG solver is implemented on the base of all the other computational kernels, which are the main blocks for AMG setup (SpMM and MWM) and for solving by PCG (SpMV). We present performance and energy consumption results obtained on IDV-A and then, to analyze the scalability potential of our main sparse linear solver based on the AMG-preconditioned Conjugate Gradient method (BCMG), we also show performance, in a weak scalability regime, obtained on the Leonardo supercomputer, whose access was granted through a Leonardo Early Access Project.

As benchmark data sets, we consider matrices and right-hand sides of algebraic systems required for the solution of the Poisson equation in 3D with homogeneous Dirichlet boundary conditions and unitary right-hand side. This is a standard benchmark test case for sparse matrix computations because it represents the computational kernel of many scientific and engineering applications, and indeed is also used in the HPCG benchmark [4]. In our case, the discretization of the problem is obtained by the classic 7-points finite-difference stencil for the left-hand side operator (the Laplacian operator), which results in a symmetric positive definite (s.p.d) matrix of coefficients well suited for PCG. Note that, in all the experiments made with the BCMG solver and discussed in the following, we stop PCG iterations when the relative residual in the Euclidean norm is less than 10^{-6} or the number of iterations reaches the maximum value fixed to 1000 (actually, in all the experiments with the linear solver, the required accuracy is obtained with no need to stop for the limit on the maximum number of iterations). In the case of the SpMV kernel, we consider, as vector operand, a vector of all ones, whereas in the case of the SpMM kernel, both the operands are the same, so that we compute the square of a Laplacian matrix. Finally, for the MWM kernel, we consider the undirected adjacency graphs of the Laplacian matrices to which suitable real weights are associated, as applied in the BCMG aggregation algorithm for the preconditioner setup.

We note that our baseline was a mono-GPU version of all the kernels, while in this project we focused on a hybrid parallel version leveraging multi-GPUs computing nodes. We are interested in analyzing both strong scalability, i.e., the reduction in the execution times when a problem with a fixed size is considered on an increasing number of parallel resources, and weak scalability properties of our kernels, i.e. when dealing with problems of increasing size, while parallel resources increase. The parallel design pattern is based on Single Program Multiple Data (SPMD) programming model relying on a row-block distribution of the system matrix and the related right-hand sides among the parallel tasks. Blocks of contiguous rows are assigned to each task. We observe that each task is associated to one GPU accelerator which oversees all the computation phases.

Details on the algorithms and parallel design patterns implemented for all kernels of the MathLib are discussed in [5] and reported in Deliverable 6.2. Here we mention, as general guidelines for users and software developers, that some approximations in original (mono-GPU) numerical algorithms have been required, so that the combination of communication-avoiding algorithms, fine-grained parallelism, and overlapping between computation and data communication could allow us to design a scalable version of the BCMG linear solver. Indeed, In the context of heterogeneous computing, where we want to use accelerators like GPUs to maximize performance, developing effective approaches for the iterative solution of linear systems brings the need for new methods, algorithms, and implementations capable of exploiting the underlying hardware and basic software. Regardless of the method adopted for solving the system, the computation on sparse matrices is particularly challenging with respect to its dense counterpart due to the irregular memory access pattern and intrinsic load imbalance caused by the sparsity pattern of the matrix rows. GPUs rely on fine-grained parallelism and access to medium size memories with high bandwidth, but also not negligible latency. Therefore, although the current software stack makes available programming environments which provide a clear interface to the features of the hardware, it is still challenging to use the above accelerators efficiently, especially for memory-bound kernels, like sparse matrix-vector products involved in iterative linear solvers. For example, the NVIDIA GPUs are built in terms of arrays of multithreaded, streaming multiprocessors, and each multiprocessor is composed of a fixed number of scalar processors. The CUDA programming paradigm is based on the concept of blocks of threads which share data. Therefore, having a regular density in the rows of a matrix is a favorable situation for high-throughput SIMD operations, whereas the irregular structure of general sparse matrices poses some limitations for efficient usage of the architecture which often are smoothed by organizing matrices in suitable data structures. Furthermore, a sequence of SIMD operations applied to the same data allows realizing the so-called data/thread-locality which makes GPU exploitation very efficient. For the same reasons, basic iterative algorithms which express a high-level of data parallelism are preferred to more complex algorithms which induce data dependency although, generally, the former may have worse convergence properties. Indeed, extra computation is often well tolerated and balanced by very efficient execution on the GPU. On the other hand, the ability to exploit the scalability of large clusters of heterogeneous nodes largely depends on appropriate coordination among multiple levels of computation so that data partitioning, data/workload balancing, data communication between GPUs, CPUs, and among distributed nodes, do not penalize in a significant way the final performance. In the paper we also included details on the algorithmic parameters which characterize the AMG preconditioner which, for the sake of brevity, we omit in this deliverable.

2.2.1 Performance and Energy Consumption on IDV-A

In this section we discuss results of different kernels, in terms of both parallel performance and power/energy consumption. For both strong scalability and weak scalability analysis we consider different matrix dimensions per each kernel, so that each GPU can be used at full load, as detailed in the tables. Power consumption measures have been obtained for all the kernels varying the number of GPUs. As also described in deliverable D6.1, our main KPIs are grouped into 3 main categories, as classified in Table 2. We point out that in this deliverable, after some discussion among the project partners, with the aim to follow a common practice, we modified KPIs for energy, adding a measure considering energy consumption and a measure whose ratio has an average power to the denominator. Finally, we observe that for all the kernels we analyze results obtained with the final configuration of IDV-A, including all the hw/sw toolchain driving the 2-phase cooling system, while for the BCMG solver we were able to make a comparison between and after the installation of the 2-phase cooling system.

KPI for energy	KPI for computational efficiency	KPI for accuracy
Iterations/Joule Iterations/AvgWatt (only for iterative linear solver)	Execution time	Yes (User's parameter dependent for iterative linear solver)
Dofs (Degrees of Freedom or unknowns)/Joule Dofs/avgWatt	(strong and weak) speedup	
	Number of iterations/time per iteration (only for iterative linear solver)	
	Accuracy	

Table 2: [MathLib CNR] KPIs

Accuracy of the results of all kernels, except the linear solver, is up to the machine precision in double precision floating-point arithmetics. In the case of the BCMG linear solver, as already said, an accuracy on the solution up to 6 digits is achieved. Different tolerances in the stopping criterion can be set up to reduce or increase solution accuracy. For the sake of completeness, we include in the tables for every kernel, also the values of peak and average power as well as the total energy measured during the execution of the kernels. Note that for the above measures we follow a so-called dynamic approach, that is we neglect power and energy of idle GPUs state and we only consider values when the kernels start their execution, including all phases of I/O from CPU to the GPU and till the GPUs go back to the idle state. In Table 3, Table 4 and Table 5 we report the performance of the SpMV, SpMM and MWM kernels, respectively.

GPUs	Dofs	Time (sec.)	Sp	Power (peak)	Power (avg)	Dofs/avgWatt	Dofs/Joule	Total Energy
Strong Scalability								
1	$3.1\text{x}10^8$	0.30	1	137	116.0	$2.7\text{x}10^6$	$3.15\text{x}10^5$	969.45
2	$3.1\text{x}10^8$	0.26	1.2	133	116.3	$2.6\text{x}10^6$	$3.13\text{x}10^5$	979.07
4	$3.1\text{x}10^8$	0.20	1.5	133	115.0	$2.6\text{x}10^6$	$3.16\ \text{x}10^5$	967.48
Weak scalability								
2	$6.2\text{x}10^8$	0.34	1.8	137	117.5	$5.2\text{x}10^6$	$7.30\text{x}10^4$	8389.12
4	$12.4\text{x}10^8$	0.48	2.5	164	117.8	$10.4\text{x}10^6$	$7.21\text{x}10^4$	16993.58

Table 3: [MathLib CNR] Performance and Energy Consumption of the SpMV kernel

We observe that, as expected for such a communication-bound kernel, parallel efficiency degrades when GPU number increase also in a weak scaling setting. Furthermore, while total energy consumption is preserved in a strong scaling setting, in the weak scaling, doubling number of Dofs going from 1 to 2 GPUs, produces an increasing ratio in energy consumption of more than 8. This increasing ratio is, as expected, about 2 going from 2 to 4 GPUs. Different behavior in terms of energy consumption is observed for the SpMM kernel, where in the strong scaling setting, the energy consumption increases for increasing GPU number both in a strong scaling and in a weak scaling setting. A large increase in energy consumption, when the number of GPUs increases as well as the number of Dofs, is also observed for the MWM kernel. The above behavior requires a deeper analysis and hopefully further tests.

GPUs	Dofs	Time (sec.)	Sp	Power (peak)	Power (avg)	Dofs/avgWatt	Dofs/Joule	Total Energy
				Strong Scalability				
1	3.8x10^7	0.33	1	179	117.0	3.3x10^5	6.75x10^4	567.09
2	3.8x10^7	0.13	2.6	122	115.5	3.3x10^5	5.08x10^4	753.80
4	3.8x10^7	0.31	1.1	122	116.5	3.3x10^5	3.35 x10^4	1143.56
				Weak scalability				
2	7.6x10^7	0.20	3.4?	131	116.5	6.6x10^5	4.19x10^4	1828.25
4	15.2x10^7	0.19	7.0?	132	116.8	13.1x10^5	4.89x10^4	3127.72

Table 4: [MathLib CNR] Performance and Energy Consumption of the SpMM kernel

GPUs	Dofs	Time (sec.)	Sp	Power (peak)	Power (avg)	Dofs/avgWatt	Dofs/Joule	Total Energy
				Strong Scalability				
1	2.7x10^8	0.257	1	198	118.0	2.3x10^6	2.94x10^5	913.72
2	2.7x10^8	0.128	2.0	160	118.5	2.3x10^6	2.95x10^5	910.65
4	2.7x10^8	0.064	4.0	141	118.0	2.3x10^6	4.06x10^5	661.35
				Weak scalability				
2	5.4x10^8	0.257	2.0	193	118.0	4.5x10^6	6.98x10^4	7685.71
4	10.8x10^8	0.257	4.0	200	118.0	9.0x10^6	9.68x10^4	11086.53

Table 5: [MathLib CNR] Performance and Energy Consumption of the MWM kernel

In the following we put results obtained with the BCMG solver before and after installation of the 2-phase cooling system. In the figures we report pictures of the power consumption behavior of the execution of the BCMG solver on 4 GPUs, when each GPU oversees a local matrix with about 61x10^6 Dofs for a linear system with a total of 2.4x10^8 Dofs.

In Figure 4 we report the behavior before the 2-phase cooling installation and in Figure 5 we show the same behavior after installation. We observe very similar power behavior but in the right side of the picture, where we see that GPUs go toward the idle state power rapidly drops in Figure 4 while in Figure 5 the same time was not sufficient to reach the idle state power, showing that the cooling controller has some impacts on the transition of the GPUs to the idle state. If we go deep inside the numbers reported in Table 6 and Table 7 we can better observe that, while performance of the solver shows a small degradation after the installation of the 2-phase cooling system, the energy consumption shows a slight reduction.

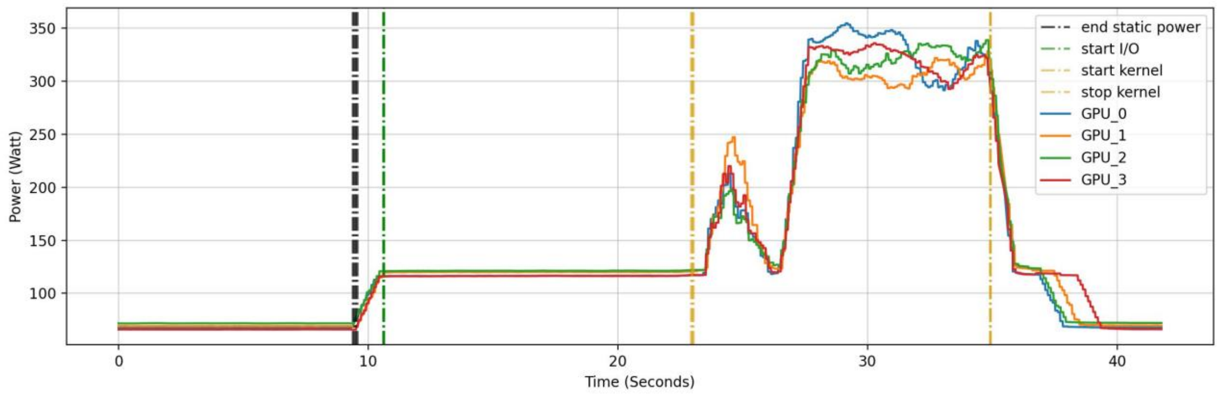


Figure 4: [MathLib CNR] Power behaviour of BCMG on IDV-A before the 2-phase cooling system installation

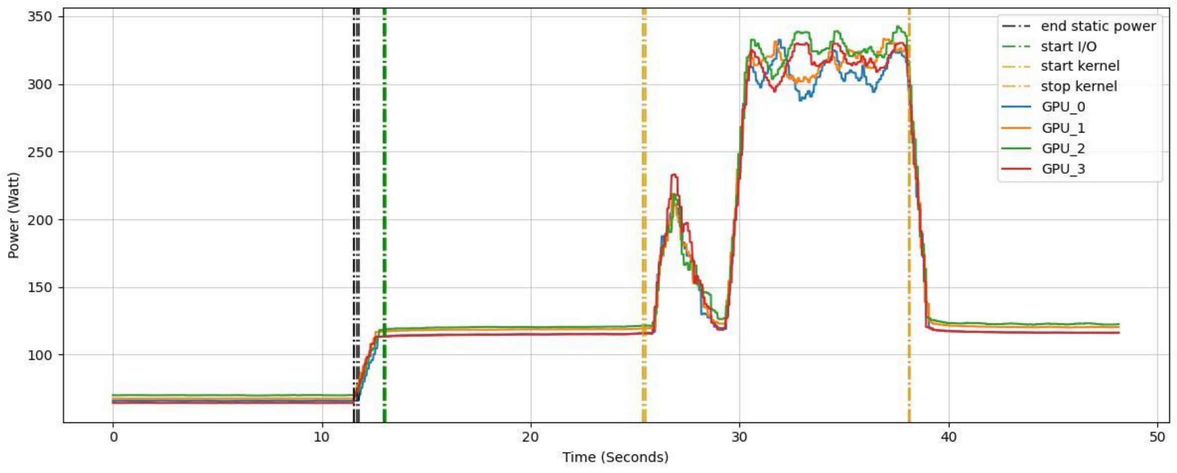


Figure 5: [MathLib CNR] Power behaviour of BCMG on IDV-A after the 2-phase cooling system installation

Table 6 and Table 7 summarize results for the BCMG solver, as for the other kernels, before and after installation of the 2-phase cooling system.

GPUs	Dofs	Time (sec.)	Sp	Power (peak)	Power (avg)	Dofs/avg Watt	Dofs/Joule	Total Energy
Strong Scalability								
1	6.1×10^{7}	8.59	1	399	178.0	3.4×10^{5}	2.85×10^{4}	2147.72
2	6.1×10^{7}	5.25	1.6	338	166.5	3.7×10^{5}	2.37×10^{4}	2578.74
4	6.1×10^{7}	4.45	1.9	247	148.3	4.1×10^{5}	1.86×10^{4}	3283.00
Weak scalability								
2	12.2×10^{7}	9.96	1.7	352	162.0	7.6×10^{5}	2.03×10^{4}	6011.36
4	24.4×10^{7}	11.27	3.1	354	153.0	16.0×10^{5}	1.90×10^{4}	12898.43

Table 6: [MathLib CNR] Performance and Energy Consumption of the BCMG solver before installation of the 2-phase cooling system

GPUs	Dofs	Time (sec.)	Sp	Power (peak)	Power (avg)	Dofs/avgWatt	Dofs/Joule	Total Energy
Strong Scalability								
1	6.1×10^7	9.77	1	400	169.0	3.6×10^5	3.18×10^4	1920.95
2	6.1×10^7	6.14	1.6	319	156.5	3.9×10^5	2.70×10^4	2268.53
4	6.1×10^7	5.68	1.7	235	140.3	4.4×10^5	2.15×10^4	2843.50
Weak scalability								
2	12.2×10^7	10.70	1.8	352	156.0	7.8×10^5	2.09×10^4	5863.95
4	24.4×10^7	11.98	3.3	342	152.5	16.0×10^5	1.95×10^4	12524.63

Table 7: [MathLib CNR] Performance and Energy Consumption of the BCMG solver after installation of the 2-phase cooling system

2.2.2 Weak Scalability results of BCMG up to 64 billion dofs on the Leonardo Supercomputer

In this section, we analyse the scalability potential of our BCMG sparse linear solver when the number of GPUs largely increases; the fixed matrix size per node is equal to 250^3=~15.6M dofs, going from 1 to 4096 GPUs of the general-purpose module of the Leonardo Italian Supercomputer operated by Cineca. Therefore, we solve problems up to 64 billion Dofs. We analyse the performance of the linear solver looking at the number of iterations to obtain the desired accuracy, the execution time to solve the system and the execution time per each solver iteration, to characterize the ability of the solver in leveraging large-scale resources for solving problems of increasing size. In the following figures, indeed, we report all KPIs which characterize the application phase of a linear solver. In Figure 6 we show the number of iterations needed to reach the required accuracy for increasing dimension and number of GPUs. We can observe that, as expected, due to a general increase in the conditioning number of the system matrices and to the uncoupled aggregation approach in the setup of the AMG hierarchy, number of iterations changes for increasing number of GPUs; after a rapid increase from 1 to 4 GPUs, we observe a good algorithmic scalability, with an almost stable number of iterations, up to 2048 GPU, while a significant increase is observed going from 2048 to 4096 GPUs showing a degradation of the quality of the preconditioner. This degradation is also responsible for the increase of the solve time, as observed from Figure 7. Moreover, we point out that we were able to solve a linear system with 64 billion Dofs in less than 3.8 seconds, with an increase in solve time of less than 6.5 times to solve a system whose size increases by more than 4 thousand times, going from 1 to 4096 GPUs. In Figure 8 we show the execution time per iteration which is a KPI showing efficiency and scalability of the implementation of BCMG and all its computational building blocks. We can observe an increase in the time per iteration which is expected for such a type of communication/memory bound problems, however we observe an increase of only 3.4 times going from 1 to 4096 GPUs, which indicates a very good implementation scalability and leads to a scaled speedup of the solve phase of 632.5 on 4096 GPUs (see Figure 9).

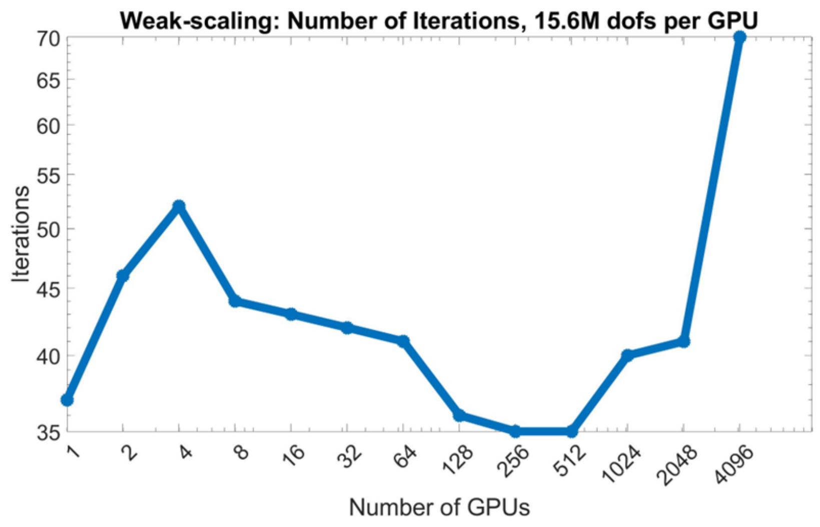


Figure 6: [MathLib CNR] Weak scalability on Leonardo up to 64 billion Dofs, number of iterations

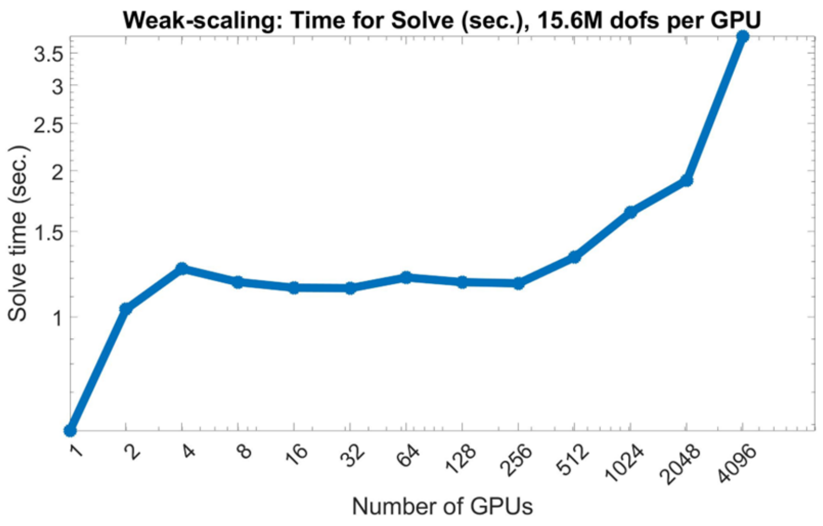


Figure 7: [MathLib CNR] Weak scalability on Leonardo up to 64 billion Dofs, solve time

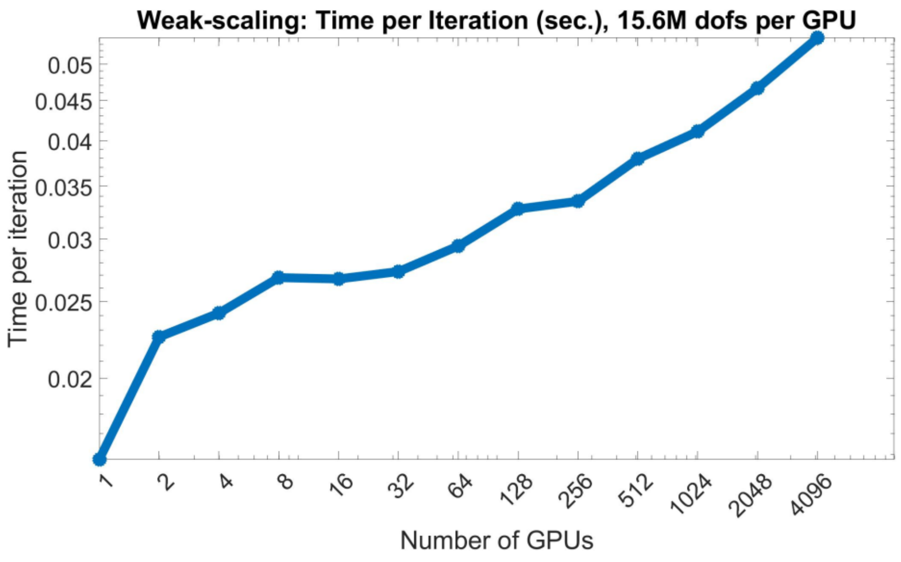


Figure 8: [MathLib CNR] Weak scalability on Leonardo up to 64 billion dofs, execution time

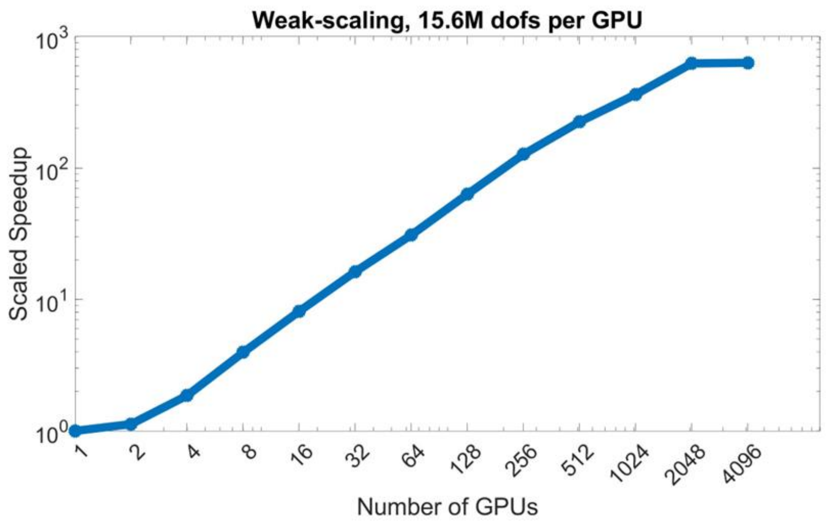


Figure 9: [MathLib CNR] Weak scalability on Leonardo up to 64 billion dofs, speedup

Comparison with NVIDIA AmgX library

In this section we show comparisons with the hybrid version of NVIDIA AmgX [6, 7], that is the state of the art for AMG-preconditioned sparse linear solver on NVIDIA GPUs. AmgX makes available various AMG preconditioners, based on different well-known coarsening approaches already available in other libraries, and producing AMG hierarchies with different computational complexities. For a fair comparison, we selected two input configurations for the AMGX solver: AMGX-C refers to a preconditioned Conjugate Gradient solver based on a robust preconditioner having very good algorithmic scalability at the cost of a large computational complexity, while AMGX-AGG refers to the PCG solver coupled with an aggregation-based preconditioner, which defines AMG hierarchies based on a similar coarsening approach and having complexities comparable with our preconditioner. Due to the limitation in using more than 128 nodes, we were able to run our tests on up to 512 GPUs of the Leonardo supercomputer. Stopping criteria are the same for all the solvers and are the same we used for the tests with BCMGX discussed in the previous sections. For these tests we were able to use a problem size per GPU equal to $200^3 = 8$ million Dofs up to more than 4 billion dofs on 512 GPUs. In Figure 10 we show number of iterations needed to reach the desired accuracy. We can see that, as expected, the best behaviour is obtained by AMGX-C, while our BCMGX shows largely better algorithmic scalability with respect to AMGX-AGG. In Figure 11 we show total solve time. Here we observe that the smallest number of iterations obtained by AMGX-C does not balance its large cost per iteration (see Figure 12) for a number of GPUs larger than 8 and then for increasing number of GPUs and problem size, our BCMGX obtains the smallest execution time with respect to the AMGX solvers. The above results are very promising and demonstrate the benefits of our new parallel algorithms and implementation design patterns for heterogeneous architectures. Our MathLib can be considered, indeed, as a main tool of the TEXTAROSSA Integrated Platform for Scientific Computing leveraging heterogeneous architectures exploiting NVIDIA GPUs.

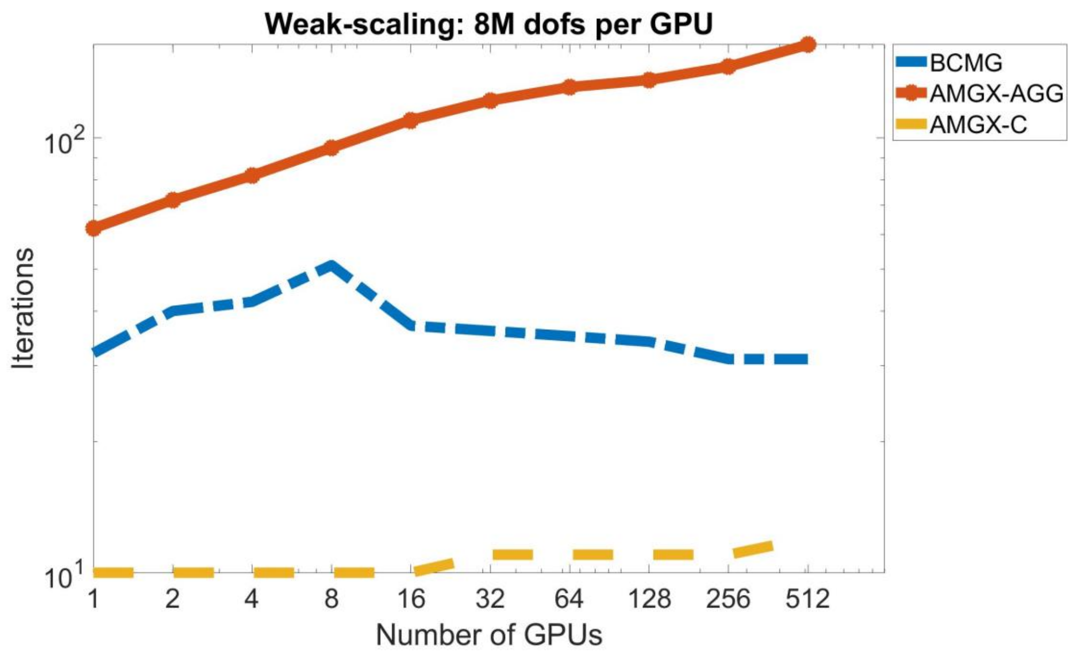


Figure 10: [MathLib CNR] Weak scalability on Leonardo. BCMGX vs AMGX, number of iterations

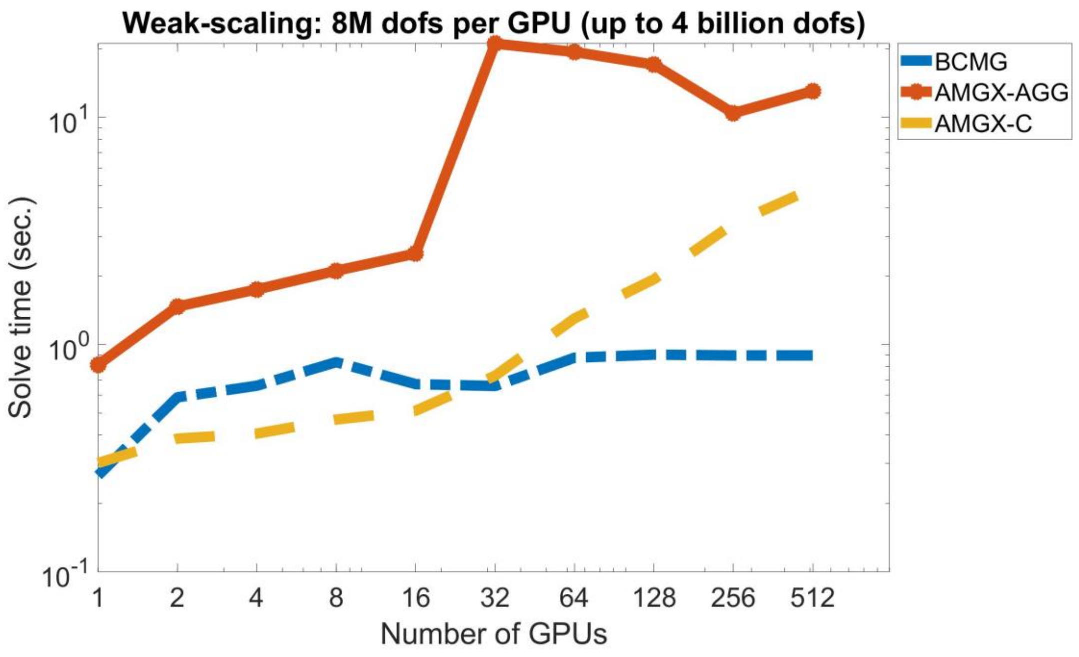


Figure 11: [MathLib CNR] Weak scalability on Leonardo. BCMGX vs AMGX, solve time.

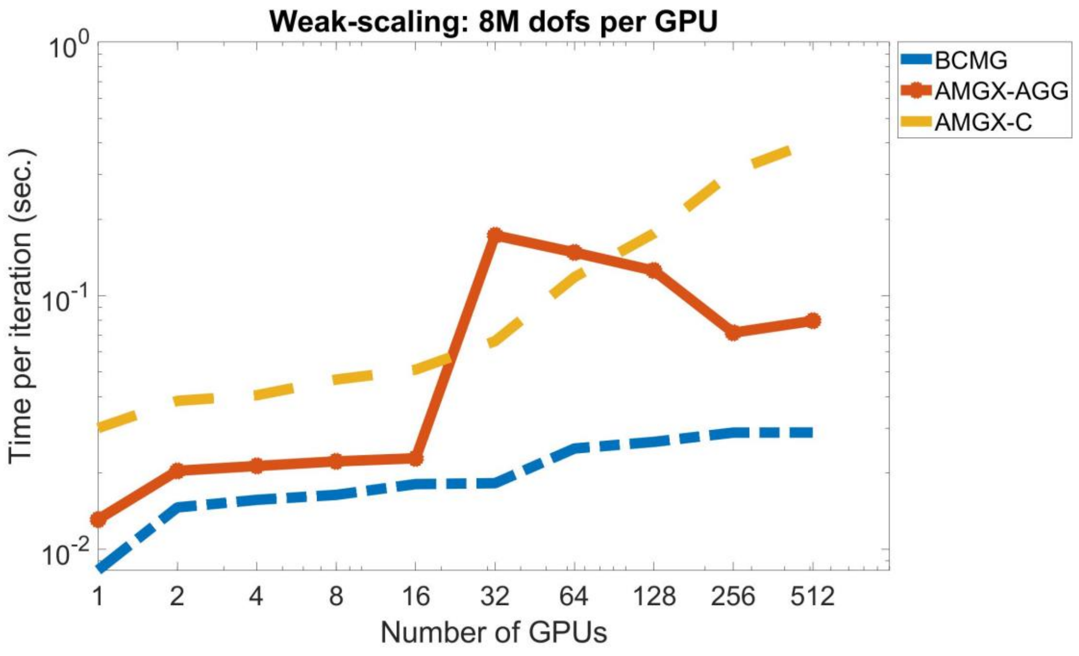


Figure 12: [MathLib CNR] Weak scalability on Leonardo. BCMGX vs AMGX, time per iteration

2.3 RTM – FRAUNHOFER

Reverse Time Migration (RTM) is used in the field of seismic for oil and gas exploration. Among other migration algorithms RTM is able to visualize more details compared with more simple algorithms as e.g. Kirchhoff migration that uses high frequency ray approximation.

Especially steep dips can't be mapped well by Kirchhoff migration. On the other hand the computational effort is higher because a simulation of the wave equation has to be done for the source and the receiver data followed by the correlation and stacking of the matching source and receiver domains (imaging condition). This means that the computational effort is especially critical with RTM algorithms.

Depending on the complexity of the velocity model RTM algorithms come at least in three flavours, the isotropic (ISO), the vertical transverse isotropy and the tilted transverse isotropy (TTI). The latter two have an anisotropy axis which can be tilted versus the vertical axis in TTI.

Especially the ISO and VTI form are typically used in practical applications. These algorithms are mostly memory bound. For this reason, increasing the pure FLOP rate is not helpful to decrease the time to solution. On the other hand, decreasing the precision of the Floating Point format might exploit better the available memory bandwidth and increase the FLOP rate under constant or even shrinking power envelope. The question to explore is if lower precision is sufficient to keep up the image quality of RTM migration algorithms calculated in full precision. As the typical Floating Point format in seismic is 32 bit IEEE Float we evaluate different 16 bit Floating Point formats. Among them 16 bit Posit format and 16 bit Floating Point format.

Previous experiments have demonstrated that RTM images calculated in reduced precision deliver competitive images compared with calculations in 32 bit Float. Simple stability tests also demonstrated positive outcome. However, these images were based on a single shot and were far from modeling a realistic seismic scenario. So the next step is to model a more realistic case. To limit the calculation time to an applicable amount we switch to 2D images and use the well known Marmousi model. The Marmousi model is an artificially created reference model based on a section in the Cuanza Basin. It is very often used in the literature to compare the performance of different migration algorithms.

Design of experiments

A small reference implementation is created that is able to calculate the isotropic RTM Marmousi example. The source and receiver wavefields can be stored in different Floating Point standards, the Float32, the Float16 and the Posit format. The RTM kernel can be calculated in the same Floating Point format matching the storage or it can be calculated in Float32 while the Floating Point format is converted back and forth. Another part is the Floating Point format of the calculated Image from the two wavefields. This format can be identical to the reduced precision Floating Point format or it can be stored in Float32. The Floating Point format of the image can be mixed arbitrarily with the calculation format of the kernel.
Furthermore the calculation of the imaging condition in reduced precision can be done including the Kahan summation. The image is calculated by calculating the correlation between the source wavefield und the receiver wavefield. The different correlations which are in the order of magnitude of the number of calculated timesteps are summed up to yield the final image. The precision of this sum can be improved using the Kahan summation. The advantage of the Kahan summation is a higher precision than using normal summation. The disadvantage of the Kahan summation is that two numbers per pixel are needed. Thus the bandwidth needed to store the image in 16 bit format is the same as the number of bit to store a 32bit image in. The Floating Point effort is even higher than calculating in Float32. However, this approach can be useful because a very lean compute core can be used to calculate the imaging condition having only ALUs in reduced precision.

Results Posit format

The Marmousi model is reduced to save computational time. The model has 240 shots that are reduced to 40 shots around the center (shots #101 to #140). Furthermore, the simulated time is reduced from 2.7 seconds to 1.62 seconds. The gird spacing is 16m in depth direction and 25m in lateral direction. The timestep is 1.405 milliseconds. The number of pixels in the domain is 185x223 (lateral times depth). The full domain would be 367x187 but is adjusted in these experiments. As only 40 shots are calculated the simulated domain can be reduced in lateral direction. In depth direction that size has been slightly extended beyond the full domain. The reason is that no specific boundary condition has been implemented. Thus, padding the domain reduces the amount of artifacts in the images created by reflected waves at the artificial boundary of the domain. The sources and receivers are coupled in 18 pixels beyond the upper boundary. In the calculation campaign different combinations of Floating Point formats in the domain, the kernel and the image are tested. All the Posit16 values use one exponent. The source signal is modeled as a Gabor wavelet.

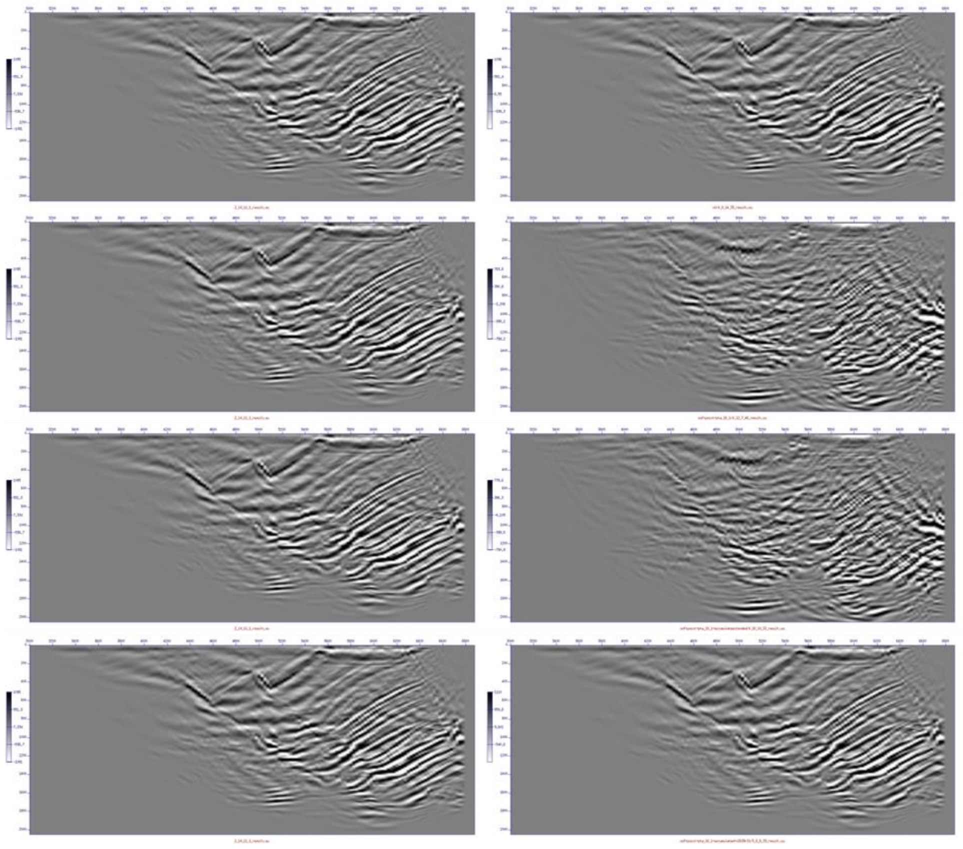


Figure 13: [RTM] Marmousi results with different combinations of Floating Point formats
Left: Reference simulation in Float32 Right: specific combination of Posit with Float32 formats
The combinations are from top to bottom:
Label "p16.1_storage_only": Domain Posit, kernel Float32, image Float32
Label "p16.1": Domain Posit, kernel Posit, image Float32
Label "p16.1_accumulate": Domain Posit, kernel Posit using quire for dot products, image Float32
Label "p16.1_accumulate_scale": Domain Posit, kernel Posit using quire while scaling model and traces, image Float32

Figure 13 and Figure 14 demonstrate that using Posit 16 with one exponent as a storage format for the domain only creates very similar images compared to the pure Float32 reference image. Using Posit for calculation of the kernel too yields unacceptable results (p16.1). The reason for the poor performance might be catastrophic cancellation in calculation of the stencil inside the kernel. To mitigate a possible cancellation a Posit quire is used to sum up the stencil. The quire has a high number of bits as an intermediate storage of the sum to avoid cancellation in intermediate results. Only the final result is rounded to the available mantissa. Image "p16.1_accumulate" demonstrates that the quire doesn't improve the image quality significantly. Another source of low precision could be a low mantissa because of large Floating Point values. The mantissa of Posit values is decreasing the more the absolute value deviates from 1. Figure 15 demonstrates the distribution of absolute values in the velocity model and the receiver data. The median of the velocity model is 2638.28. The average of the two medians (negative values and positive values) is 30.981. The receiver data and the velocity data is scaled so that the median values map to 1. The output images are scaled back so that the scaling is cancelled out. Simulation "p16.1_accumulate_scale" demonstrates that the scaling fixes the broken image. In Figure 13 simulation "p16.1_scale" demonstrates that the scaling alone is sufficient to create a good image and that the quire is not needed. Simulation "p16.1_scale_img_p16.1" and "p16.1_scale_img_p16.1_kahan" demonstrate that an image stored in Float16 doesn't make the image unusable

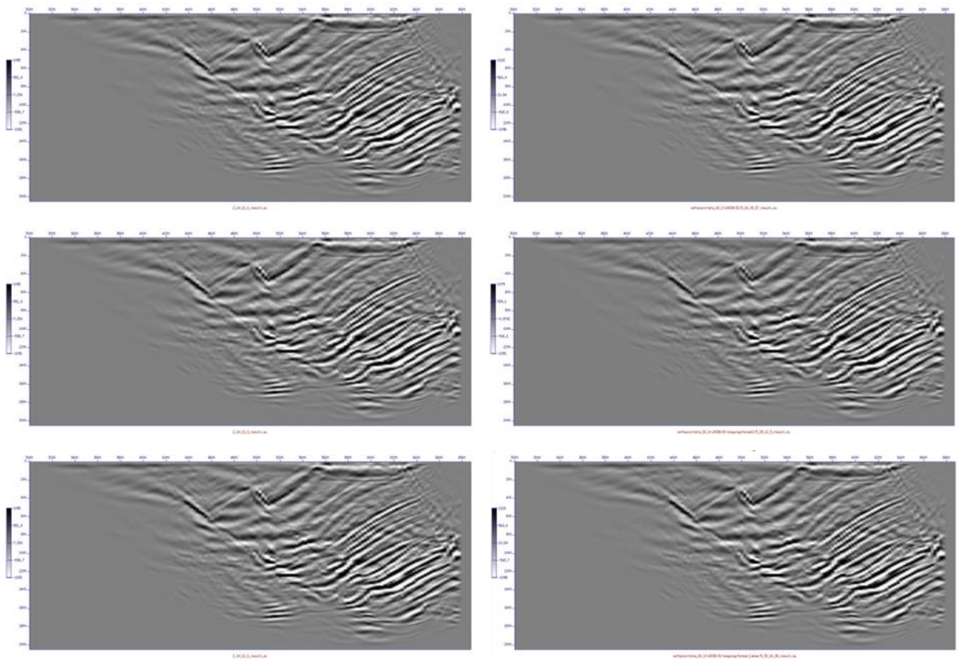


Figure 14: [RTM] Marmousi results with different combinations of Floating Point formats continued
Left: Reference simulation in Float32 Right: specific combination of Posit with Float32 formats
The combinations are from top to bottom:
Label "p16.1_scale": Domain Posit, kernel Posit scaling model and traces, image Float32
Label "p16.1_scale_img_p16.1": Domain posit, kernel Posit scaling model and traces, image Posit
Labe l"p16.1_scale_img_p16.1_kahan": Domain posit, kernel Posit scaling model and traces, image Posit Kahan

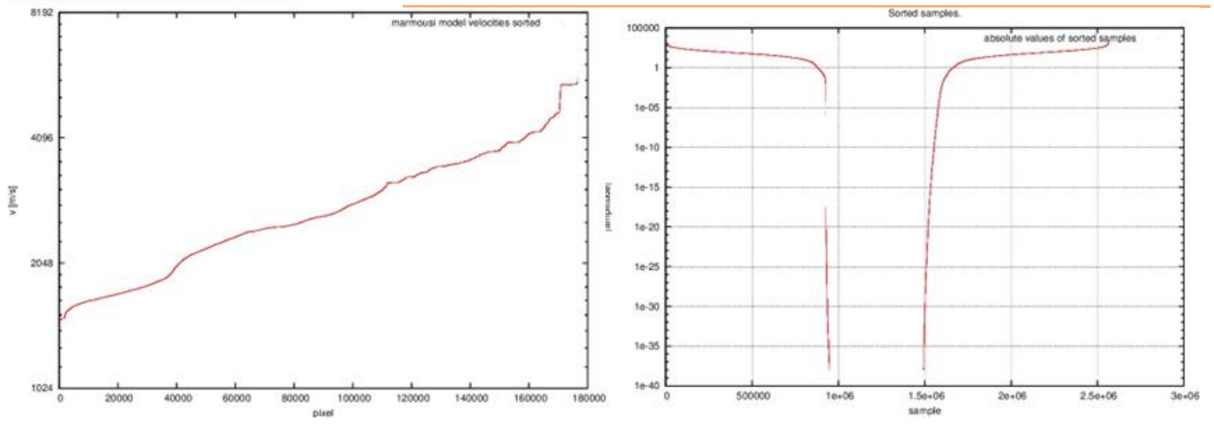


Figure 15: [RTM] Distribution of absolute values in velocity model (left) and all the receiver traces (right)
Input data is sorted ascending for size and plotted as absolute value.

Figure 16 plots the infinity norm of the difference between Float32 and Posit normalized by the infinity norm of the Posit image. The simulation using Posit as storage format for the image deviates significantly more from the reference than all the other useful cases. The usage of Kahan summation brings back the deviation to similar values as other useful cases (approx. 4%).

To have close to zero difference between Float32 and Posit is not necessarily crucial. Most important is to have no shifts of the reflectors especially in depth direction. On the other hand a difference close to zero implies few shifts. But although the case ”p16.1_scale_img_p16.1” has significantly higher deviation towards Float32 as e.g. “p16.1_scale” the eye cannot see relevant shifts of reflectors in both cases.

2.3.1.1 Numerical Stability

To save computational time the simulation has been restricted to a subset of the model. Especially the simulated time has been reduced. To verify that the Marmousi example is stable also for longer simulated times the simulated times was increased in a further reduced test example. Here the development of the total energy of the source wavefields is plotted over time. A proxy for the total energy is the summed squares of all the pixels. According to the physics a conservation of energy is expected over time. So this property is a must have of the algorithm and a violation of this property reveals numerical instability.

Figure 17 demonstrates a significant increase of the energy beyond 1.6 seconds simulated time. The same simulation with 17 bit Posits demonstrates that this explosion of energy can be avoided by using a slightly higher precision. This means that the precision of the Posit 16 bit is very close to sufficient to avoid this explosion. In the end a reorganization of the kernel was sufficient to stabilize the algorithm even if 16 bit Posits.

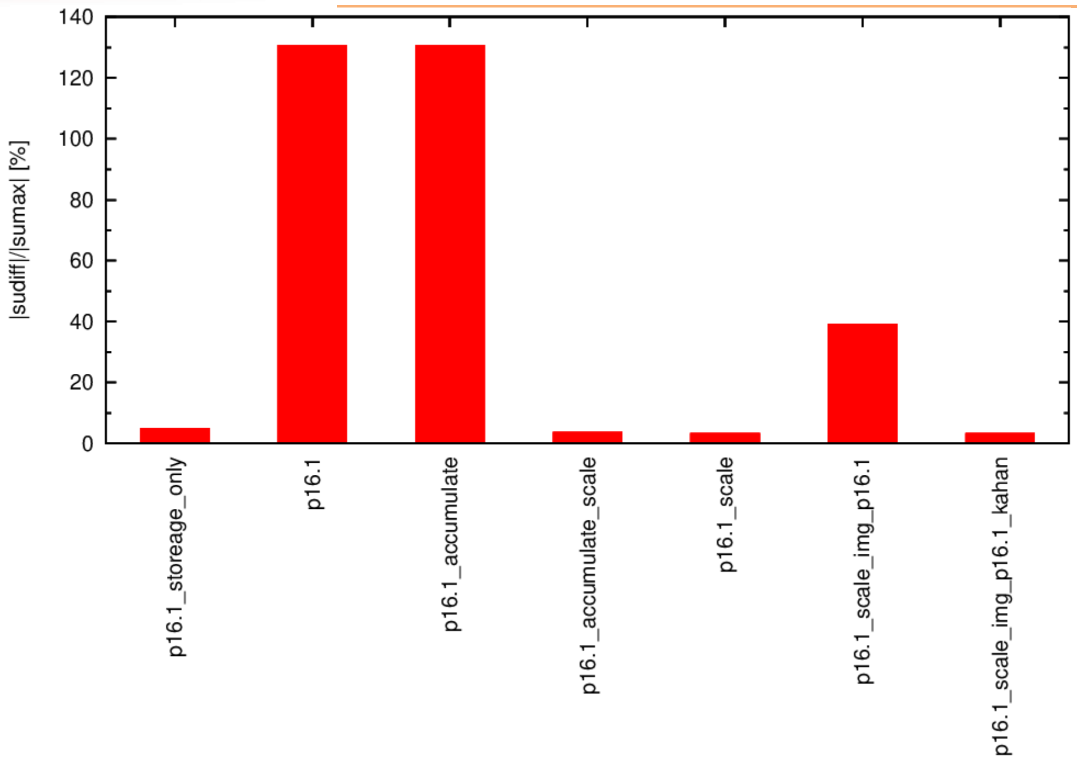


Figure 16: [RTM] Distribution of absolute values in velocity model (left) and all the receiver traces (right)

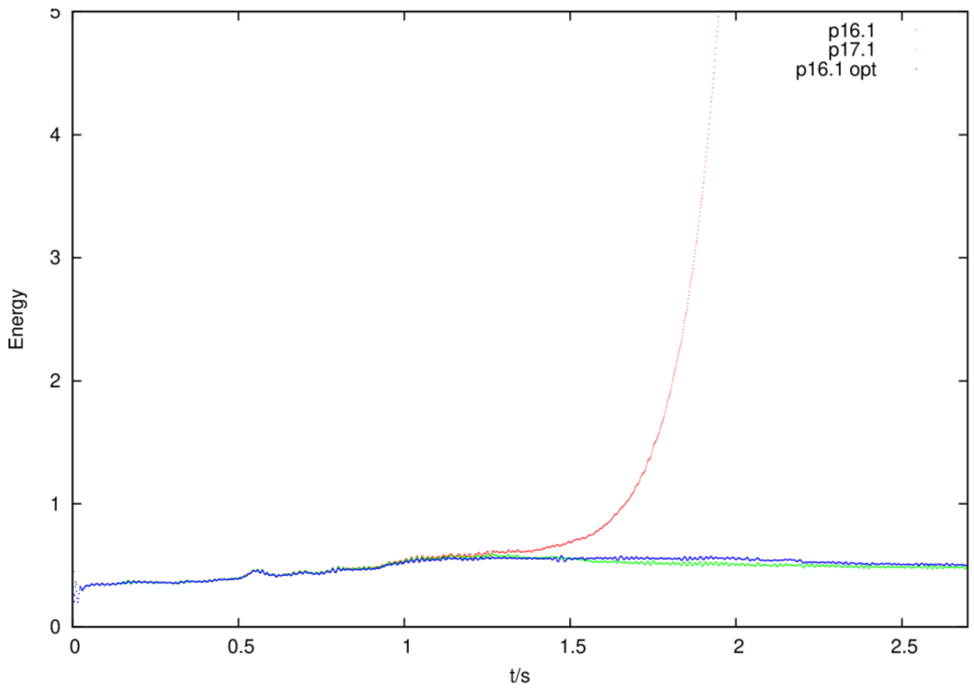


Figure 17: [RTM] Distribution of absolute values in velocity model (left) and all the receiver traces (right)
The energy of shot number 120 of 240 in the Marmousi model is plotted over time. The Floating Point format is Posit with 16 bit (red), 17 bit (green) and 16 bit with reorganized kernel (blue). Always one exponent is used. Parameters: Grid spacing is 25x16 meters, timestep is 1.405ms, the simulated time is from 0 to 2.7s. The number of pixels is 185x223 for the red and the green curve and 185x373 for the blue curve.

2.3.1.2 Conclusion

To use the Posit 16 one exponent Floating Point to store the source and receiver wavefields delivered acceptable images more or less out of the box. Using the same format for calculation of the kernel raised some issues. These issues could be solved by scaling the model and the receiver samples combined with a numerical reorganization of the kernel. Using the Posit as format to aggregate the correlated images too increases the deviation towards Float 32 bit significantly but the results are still acceptable. Additionally, the Kahan summation can be used to rebuild a low deviation towards Foat32. However, this mitigates the bandwidth savings of the reduced precision format. On the other hand, this is only a minor drawback as the number of memory accesses to the image during the stacking process is small compared to the memory accesses to the wavefields during the stacking process and during the kernel calculation. The advantage to use Kahan summation might be that the compute core can be designed more simply by dropping the support for Float32 completely.

Results Float16 format

The simulation in Float 16 bit Floating Point format is less compute time consuming so that the full Marmousi model can be calculated. The gird spacing is 16m in depth direction and 25m in lateral direction. The simulated time is 2.7 seconds, the timestep is 1.405 milliseconds. The number of pixels in the domain is 439x373 (lateral times depth). The full domain would be 367x187 but is padded to avoid artifacts caused by reflections on the artificial boundaries. The sources and receivers are coupled in at 93 pixels beyond the upper boundary. Sources and Receivers are scaled to avoid overflow of exponents in the Float 16 data type. The maximum representable value is 65504 in Float16. In the calculation campaign different combinations of Floating Point formats of the domain, the kernel and the image are tested. The source signal is modeled as a Gabor wavelet.

Figure 18 depicts the calculated images. The first model “f16” shows storage and calculation of the wavefields in Float16 while the image is stacked in Float32. The next case uses Float 16 bit for stacking of the image too while the case “f16_img_f16_kahan” uses Kahan summation additionally to stack the images. The remaining two cases store the wavefields in -Float16 and calculate the kernel in Float32. The “f16_storage_only” case stacks the image in Float32 while the “f16_storage_only_img_f16” stacks the image in Float16. All the images show an acceptable quality. The eye can detect very subtle differences between images where the kernel has been calculated in Float32 compared to kernels calculated in Float16.

Figure 19 plots the relative norm of difference versus the pure Float32 case as reference. The cases where the kernel was calculated in Float16 are all very similar around 7% deviation. The deviation of the respective Posit16 cases towards Float32 (3%) was lower (better). Calculating the kernel in Float32 reduces the deviation to 0.5% or 4% if the image is stacked in Float16. Stacking the image in Float16 makes a significant difference here.

2.3.1.3 Numerical stability

To verify that the Marmousi example is stable while calculated in Float16 the development of the total energy of the source wavefields is observed over time (0s-2.7s). A proxy for the total energy is the summed squares of all the pixels. In none of all the cases the plotted energy over simulated time did show any explosive increase (data not shown). So no signs of instability have been detected.

2.3.1.4 Conclusion

All the different combinations of Float16 with Float32 delivered acceptable images and numerically stable simulations. If the kernel was calculated in Float16 the deviation versus the reference was double compared to the respective Posit16 calculations. On the other hand, if the kernel was calculated in Float32 the Float16 simulation had smaller deviations towards the reference. Scaling the input data was necessary with Float16 as well as with Posit16.

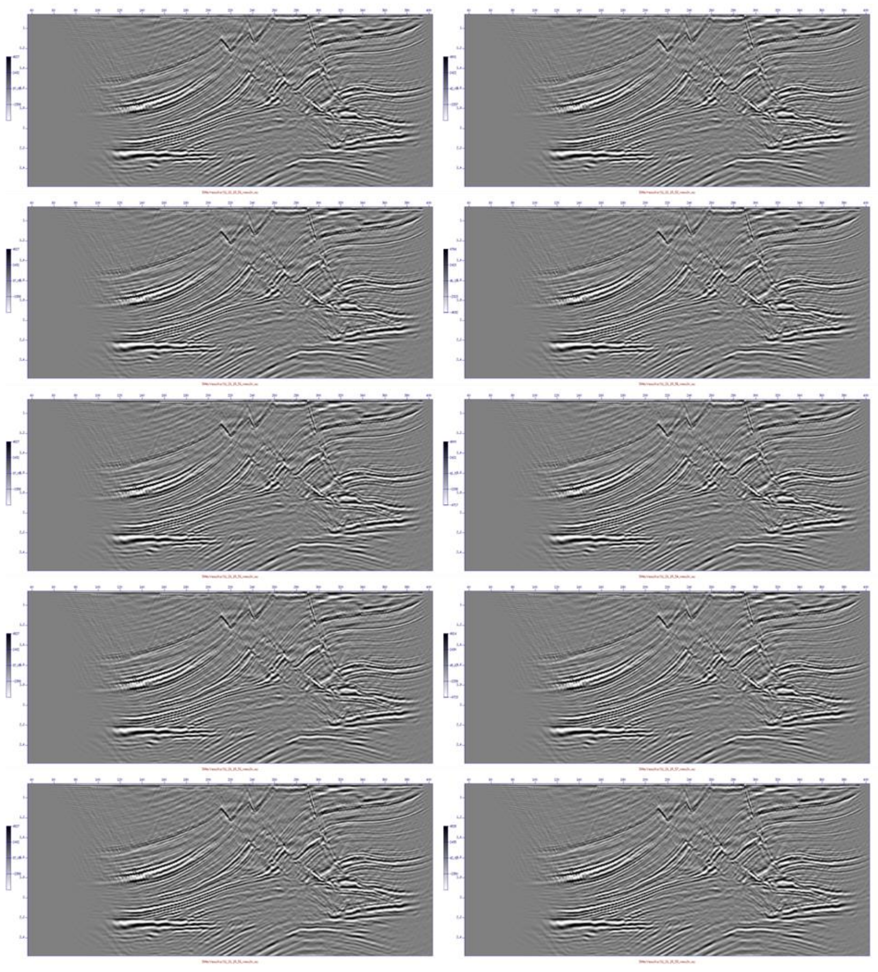


Figure 18: [RTM] Marmousi results with different combinations of Floating Point formats
Left: Reference simulation in Float32 Right: specific combination of Float16 with Float32 formats
The sources and receivers are scaled to avoid exponent overflow in Float16 values.
The combinations are from top to bottom:
Label “f16”: Domain Float16, kernel Float16, image Float32
Label ”f16_img_f16”: Domain Float16, kernel Float16, image Float16
Label ”f16_img_f16_kahan”: Domain Float16, kernel Float16, image Float16 with Kahan summation
Label “f16_storage_only”: Domain Float16, kernel Float32, image Float32
Label “f16_storage_only_img_f16”: Domain Float16, kernel Float32, image Float16

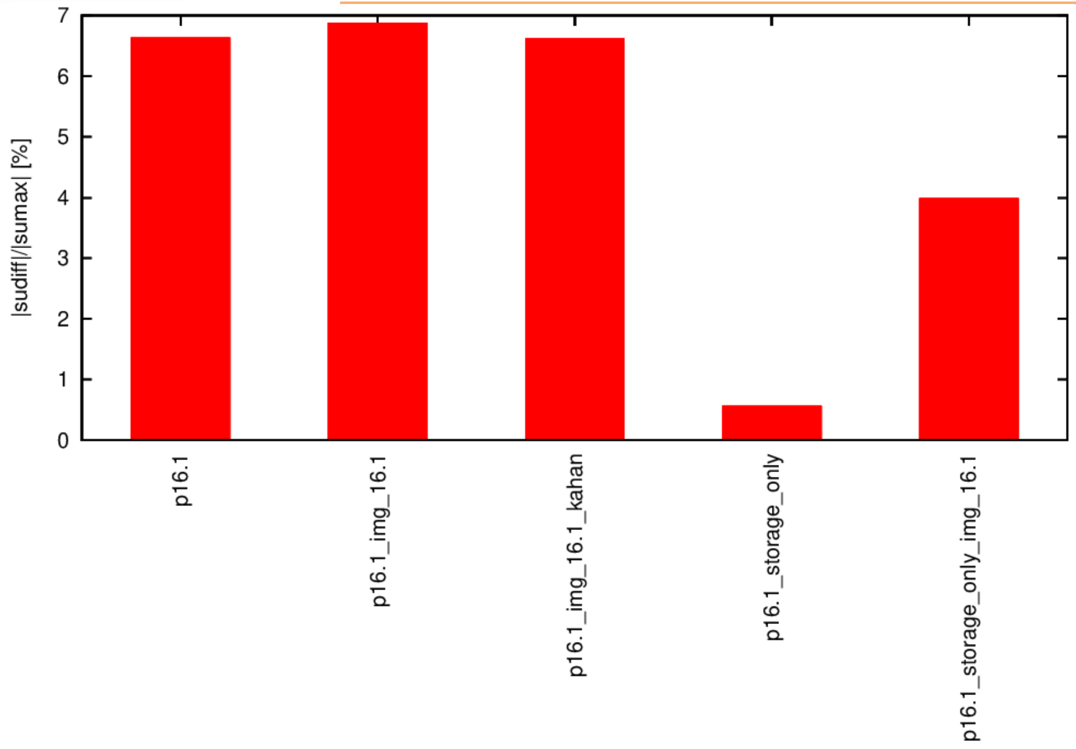


Figure 19: [RTM] Infinity norm of difference between Float16 and Float32 as percentage of the infinity norm of the Float16 example

2.4 HEP

High-Energy Physics (HEP) experiments are increasingly using a mix of processing hardware (heterogeneous architectures) to keep up with computing solutions offered by accelerator technologies.

The TEXTAROSSA project focused on two applications from CERN's Patatrack team: Pixeltrack (a pixel tracking algorithm used for CMS detector) and CLUE (a cluster algorithm for high-granularity calorimeters). Our first goal was to rewrite them using a portability layer. This allows a single codebase to run efficiently on various hardware types from different vendors. Recoding for each specific architecture is impractical.
Following objective was to evaluate the performance of these applications in two key areas:

- throughput: number of reconstructed events per second;
- energy efficiency: number of reconstructed events per Joule, obtained from the ratio between throughput and power.

As reported on deliverable D6.2, for the TEXTAROSSA project we concluded that while SYCL is a promising approach, Intel's oneAPI implementation isn't stable or mature enough for our needs. Specifically, we require running the applications used to collect and process data from HEP detectors on various mixed hardware platforms. For that reason, we opted to use a different abstraction layer called Alpaka [1] to collect data for the TEXTAROSSA project. As quoted from the Alpaka documentation: "The Alpaka library is a C++14 header-only abstraction library designed for accelerator development. Its goal is to ensure performance portability across accelerators by abstracting (not hiding!) the underlying layers of parallelism."

While coding with Alpaka is more complex than with SYCL, this library supports a wider range of devices from different vendors compared to oneAPI, making it better suited for our project.
Therefore, we opted to conduct all measurements leveraging the Alpaka version of the application, in this way we were able to run the same source code on various devices without any modifications. We used the applications to characterize the IDV-A (CPU only, CPU+GPU) and IDV-E (CPU only) node's performance.

2.4.1 Baseline KPIs assessment on Dibona node

We repeated the tests and measurements on the Dibona node, that were already reported in Deliverable 6.2, to be more confident in assessing the baseline for the applications' KPIs to be used in the comparison with the IDV-A and IDV-E nodes' performance.

2.4.1.1 Tests on CPU

Our focus on the CPU is twofold: firstly, to verify that our code efficiently scales increasing the number of processing cores. Secondly, we want to see how energy efficiency is affected by this scaling. We achieved this by running both applications with a varying number of cores and threads. To measure power consumption, we employed the *likwid-perfctr* tool. This tool also assisted us in accurately specifying the number of cores to be used.
The number of threads could be indicated in the command we use to run the application, instead. In the same command we also specified how long the applications must run. Every test was 2-minute long and the likwid-perfctr tool extracted the power consumption values every second. An example of one of those execution command is:

```
likwid-perfctr -f -C S<socket_id>:<cores> -g ENERGY -t 1s -O -o
<output-file>.csv ./alpaka --serial --runForMinutes 2 --numberOfThreads
<threads>
```

The results we achieved are reported in the section below. Table 8 and Table 9 show the measures of throughput (events/second), Power (W) end Energy Efficiency (a.u., normalizing to 1.0 the events/J obtained on a single core) for the CLUE application and for the Pixeltrack application when they run on CPU, scaling the number of logic cores.

CLUE on CPU				
cores	**thread**	**Throughput (events/sec)**	**Power (W)**	**Energy efficiency (ref single core)**
1	1	4.341	81.114	1.0
2	2	8.574	85.074	1.748
4	4	16.878	92.266	3.077
8	8	33.720	107.233	5.169
16	16	66.586	136.053	8.049
24	24	98.840	166.268	9.676
48	48	189.239	162.402	18.980
64	64	232.670	172.709	21.917
72	72	253.44	181.032	22.777

96	96	296.225	173.950	27.745

Table 8: [HEP] KPIs for CLUE on Dibona CPU

Pixeltrack on CPU				
cores	**thread**	**Throughput (events/sec)**	**Power (W)**	**Energy efficiency (ref single core))**
1	1	30.118	82.116	1.0
2	2	60.7018	86.439	1.890
6	6	178.051	103.708	4.354
12	12	350.269	129.147	6.835
24	24	705.473	182.601	9.556
48	48	1365.25	177.424	13.306
64	64	1573.48	190.017	20.419
72	72	1604.44	192.939	20.535
96	96	1353.11	178.625	18.733

Table 9: [HEP] KPIs for Pixeltrack on Dibona CPU

The scaling of throughput for both applications is illustrated on Figure 20 and Figure 21.

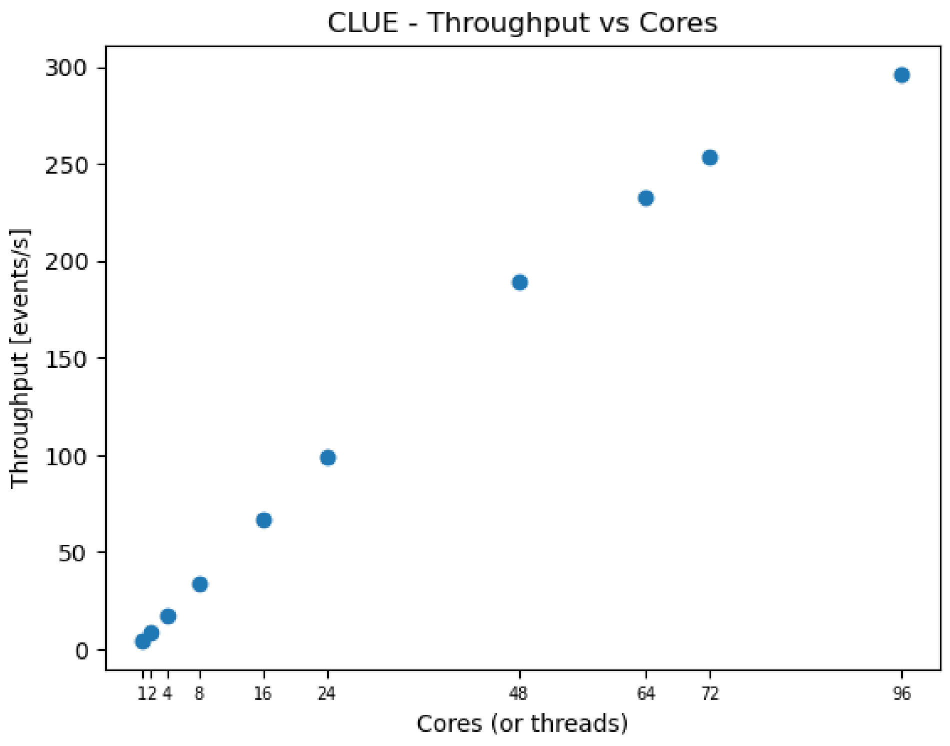


Figure 20: [HEP] Throughput vs cores for CLUE on Dibona CPU

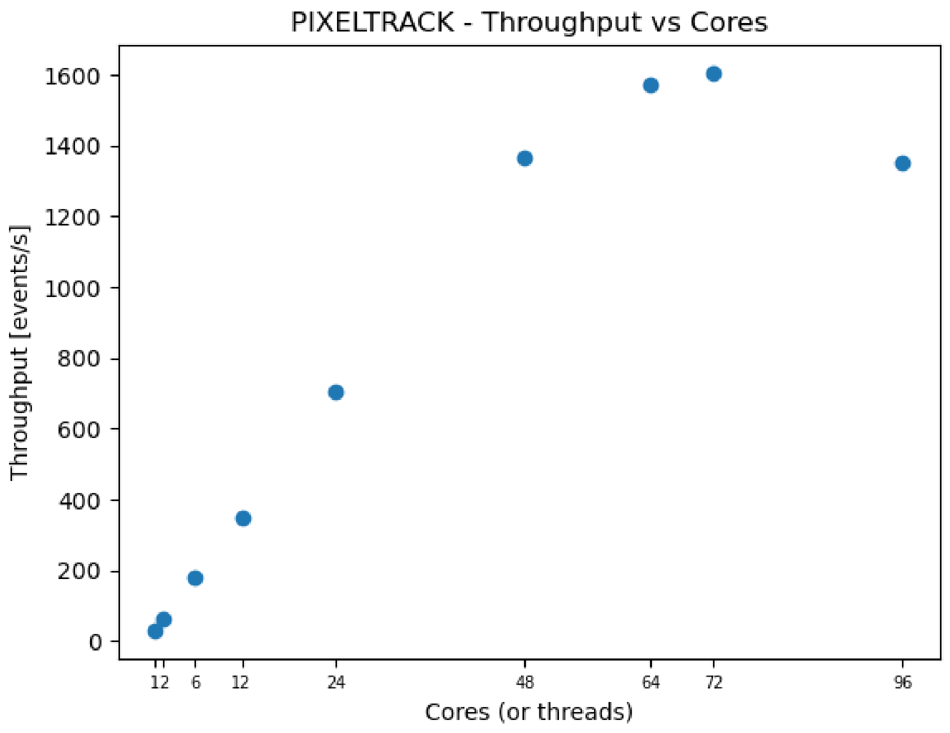


Figure 21: [HEP] Throughput vs cores for Pixeltrack on Dibona CPU

Figure 22 and Figure 23 explore the relationship between the number of cores used and the applications' energy efficiency. It's important to note that the data in both these figures is normalized, meaning it's been adjusted relative to the performance achieved with a single thread on a single core.

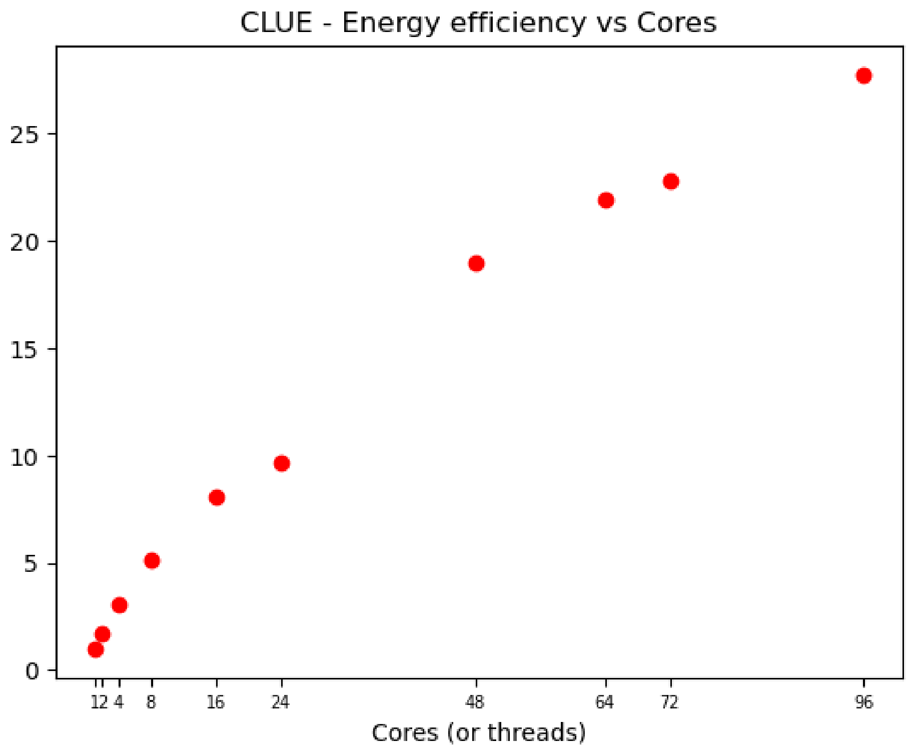


Figure 22: [HEP] Energy efficiency vs cores for CLUE on Dibona CPU

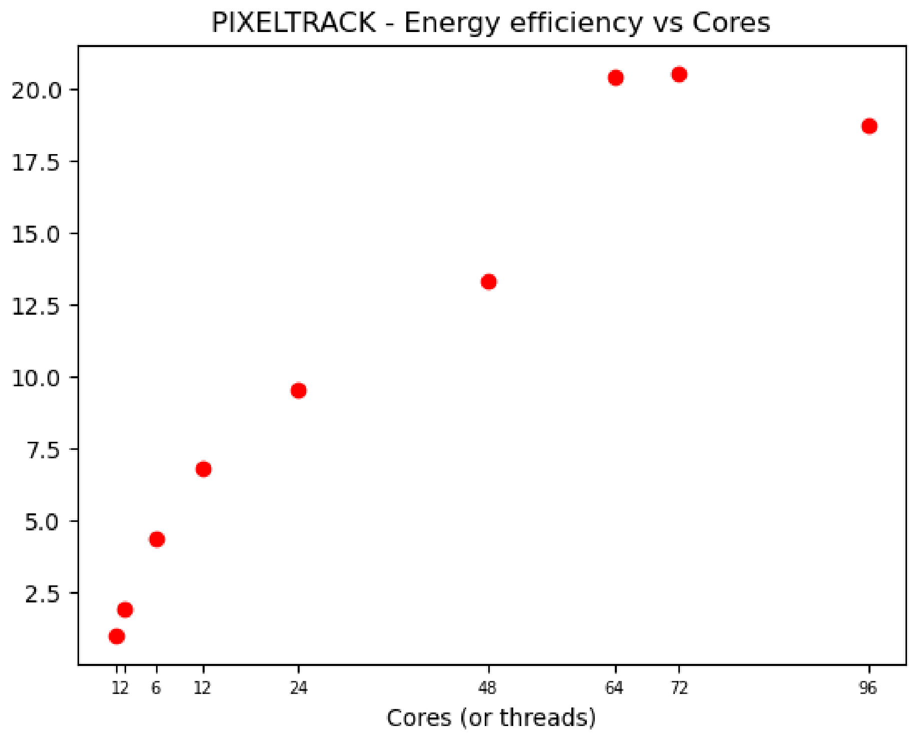


Figure 23: [HEP] Energy efficiency vs cores for Pixeltrack on Dibona CPU

Figure 20 and Figure 21 show that the performance grows linearly when we run one thread per core using one socket. This trend remains almost constant for both the applications, but for the Pixeltrack algorithm, above 64 threads, the performance gets worse. A similar behaviour can be noticed in Figure 22 and Figure 23, showing the scaling of the Energy Efficiency KPI with the number of CPU cores used by the applications.
This difference might be caused by the Pixeltrack implementation possibly requiring more memory accesses, incurring in resource contention.

2.4.1.2 Tests on GPU

Our focus on the GPU is similar to the CPU analysis: demonstrating how well our code utilizes an increasing number of GPUs and how energy efficiency is affected.
To achieve this, we fixed the number of CPU threads per GPU at 12, ensuring each thread runs on a separate core. This ensures all 48 physical cores are utilized when using 4 GPUs. As with the CPU tests, the number of CPU threads can be specified in the application's launch command. Additionally, this command allows configuring the mapping between GPUs, CPU threads, and cores. All our tests ran for a duration of 2 minutes.
For illustration purposes, here's an example command used to run the application on 2 GPUs:

```
(CUDA_VISIBLE_DEVICES=0 taskset -c 0-11 ./alpaka --cuda --runForMinutes 2 -
-
numberOfThreads 12) & (CUDA_VISIBLE_DEVICES=1 taskset -c 12-23 ./alpaka --
cuda -
-runForMinutes 2 --numberOfThreads 12)
```

For the power consumption measurements, we use the NVIDIA-smi tool; we read the value from the GPUs every 5 seconds to not affect their performance. As example:

```
NVIDIA-smi dmon -i <id_gpus> -d <time>
```

The results we achieved are reported in the section below. Table 10 and Table 11 show the measures of throughput (events/second), Power (W) and Energy Efficiency (events/J) for the CLUE application and for the Pixeltrack application when they run on one or more GPUs.

CLUE on GPU – A100			
GPU	**Throughput (events/sec)**	**Power (W)**	**Energy efficiency (events/J)**
1	1401.940	175.500	7.991
2	2937.820	368.261	7.979
3	4369.630	551.833	7.922
4	5651.190	728.849	7.756

Table 10: [HEP] KPIs for CLUE on GPU on Dibona node

Pixeltrack on GPU – A100			
GPU	**Throughput (events/sec)**	**Power**	**Energy efficiency**

		(W)	(events/J)
1	2174.430	149.333	14.59
2	4302.790	300.804	14.34
3	6370.730	445.749	14.29
4	8591.570	595.272	14.43

Table 11: [HEP] KPIs for Pixeltrack on GPU on Dibona node

Figure 24 and Figure 25 show for both applications the close-to-perfect linear scaling of throughput KPI with the number of the used GPUS on the Dibona node.

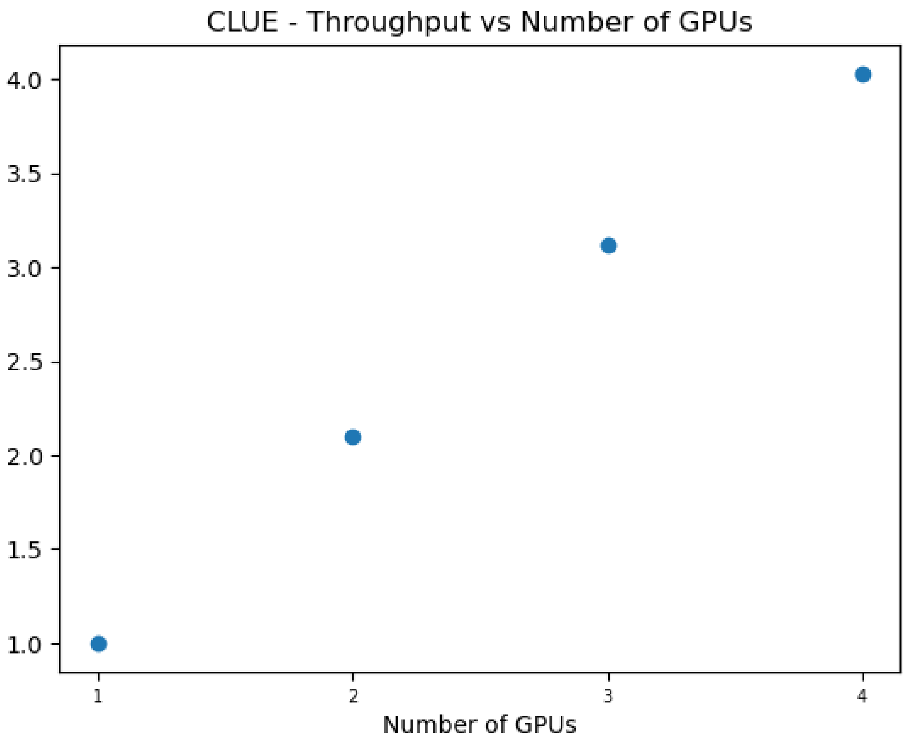


Figure 24: [HEP] Throughput (a.u.) vs number of used GPUs for CLUE on Dibona node

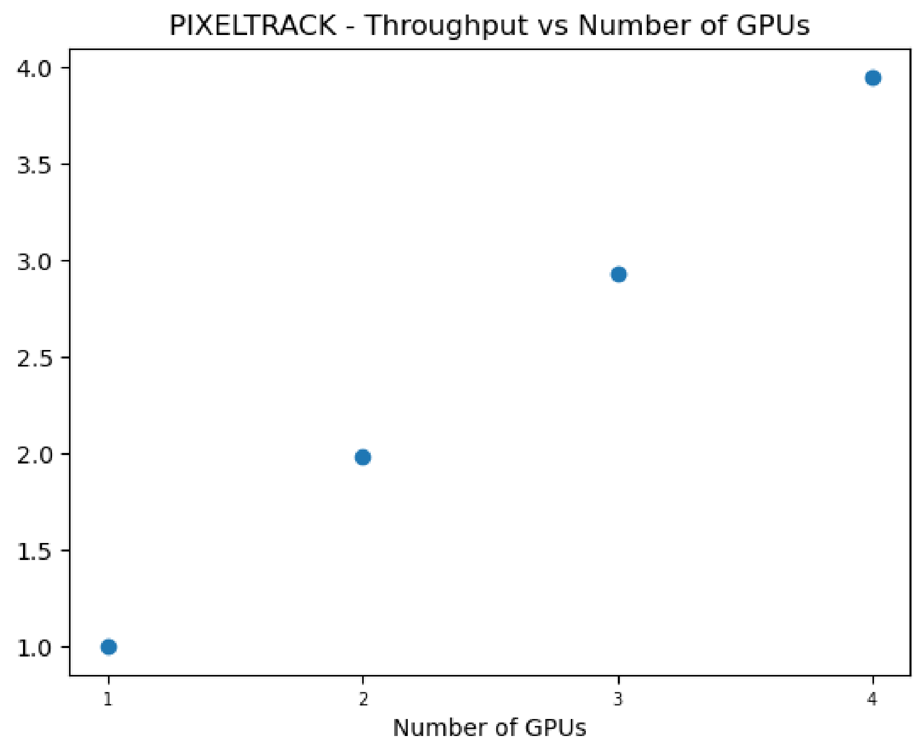


Figure 25: [HEP] Throughput (a.u.) vs number of used GPUs for Pixeltrack on Dibona node

Figure 26 and Figure 27, on the other hand, show for both applications an almost constant trend of the energy efficiency KPI (normalized to what is obtained on one GPU) scaling the number of the used GPUs on the Dibona node.

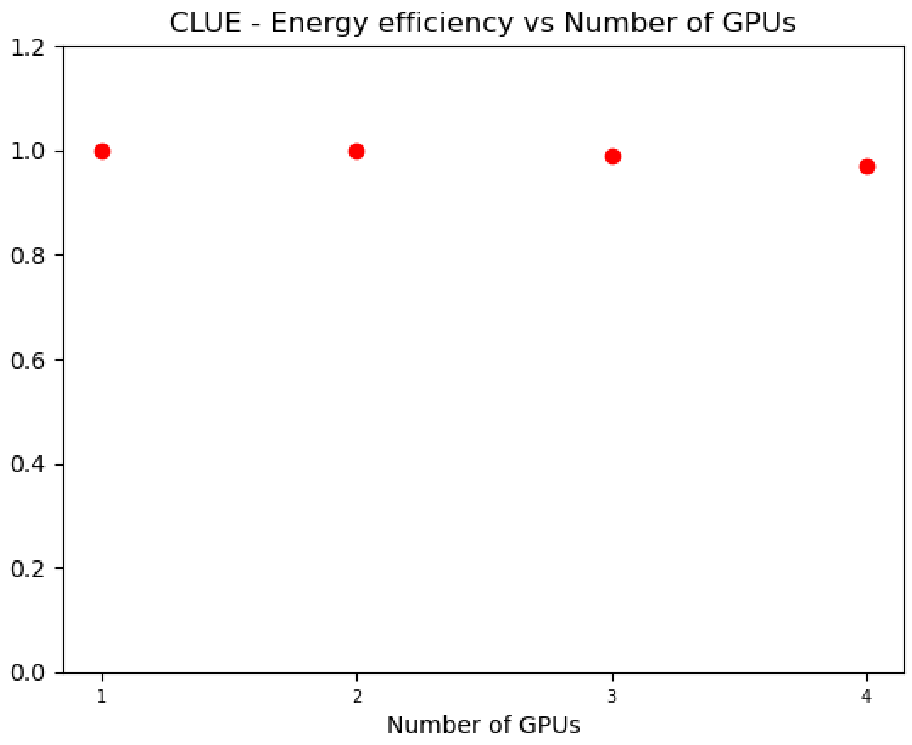


Figure 26: [HEP] Energy efficiency (normalized to one GPU) vs number of used GPUs for CLUE on Dibona node

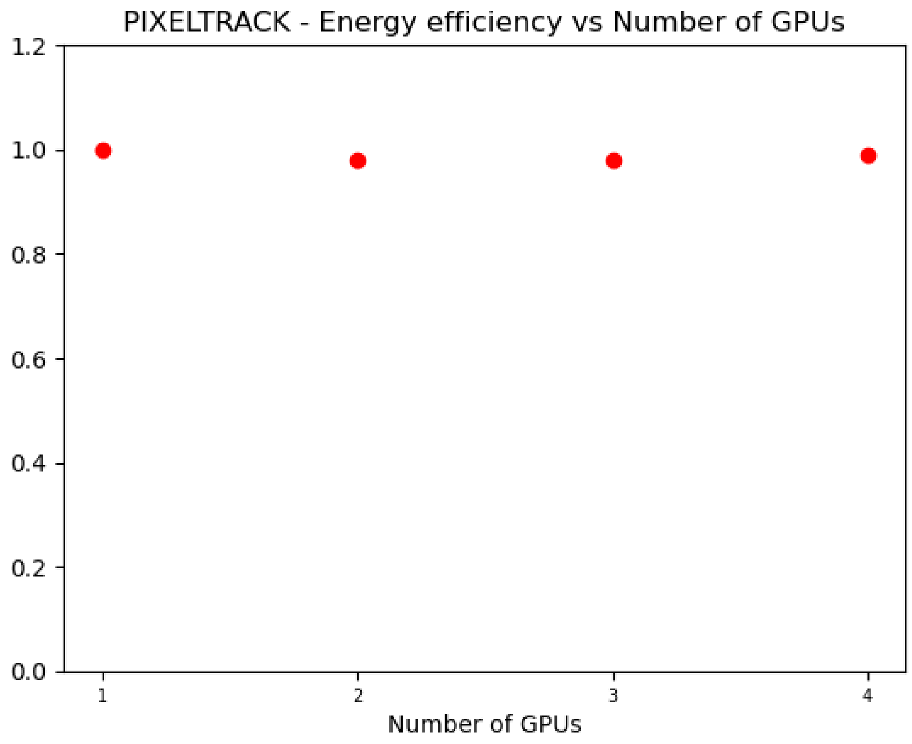


Figure 27: [HEP] Energy efficiency (normalized to one GPU) vs number of used GPUs for Pixeltrack on Dibona node

2.4.2 Tests on IDV-A

Being available the IDV-A, a dual-socket server equipped with two Intel(R) Xeon(R) Platinum 8470 (Sapphire Rapids generation with 52 *Physical* cores Total Threads.) and 4 NVIDIA H100-64MB, we had the chance to test the performance of the CLUE and Pixeltrack benchmarking applications on this node implemented with more recent technologies. In this section we report the results of the measured applications' KPIs on the IDV-A node, for both CPU only and CPU+GPU configurations.

2.4.2.1 Tests on CPU

In Table 12 and Table 13 the measurements of throughput (events/second), Power (W) and Energy Efficiency (a.u.) for the both CLUE and Pixeltrack applications run on their CPU versions are listed.

CLUE on CPU				
cores	**thread**	**Throughput (events/sec)**	**Power (W)**	**Energy efficiency (ref single core)**
1	1	3.198	405.81	1.0
2	2	6.364	408.87	1.975
6	4	18.865	417.10	5.740
12	8	37.462	428.19	11.102
24	16	75.028	451.89	21.068
52	52	161.582	504.82	40.616
64	64	196.779	529.61	47.148
72	72	220.188	546.27	51.148
96	96	314.112	607.79	65.580

Table 12 [HEP] KPIs for CLUE on CPU for IDV-A node

Pixeltrack on CPU				
cores	**thread**	**Throughput (events/sec)**	**Power (W)**	**Energy efficiency (ref single core)**
1	1	21.469	407.23	1.0
2	2	44.915	409.62	2.080
6	6	132.096	420.32	5.961
12	12	261.728	431.91	11.494
24	24	527.451	466.24	21.458
52	52	1138.95	535.87	40.316
64	64	1138.95	568.78	45.916
72	72	1528.82	591.81	49.001
96	96	2060.88	672.98	58.086

Table 13: [HEP] KPIs for Pixeltrack on CPU for IDV-A node

Figure 28 and Figure 29 show, as in the case of the Dibona node, a close-to-perfect linear trend of the throughput KPI with the scaling of the number of used CPU cores, with a marginal improvement in absolute terms on IDV-A for this KPI for the CLUE application, while there is a noticeable improvement in scaling approaching the maximum number of available cores for the Pixeltrack application, with a better scaling behaviour on IDV-A and a noticeable improvement in the maximum attainable throughput in the new architecture.

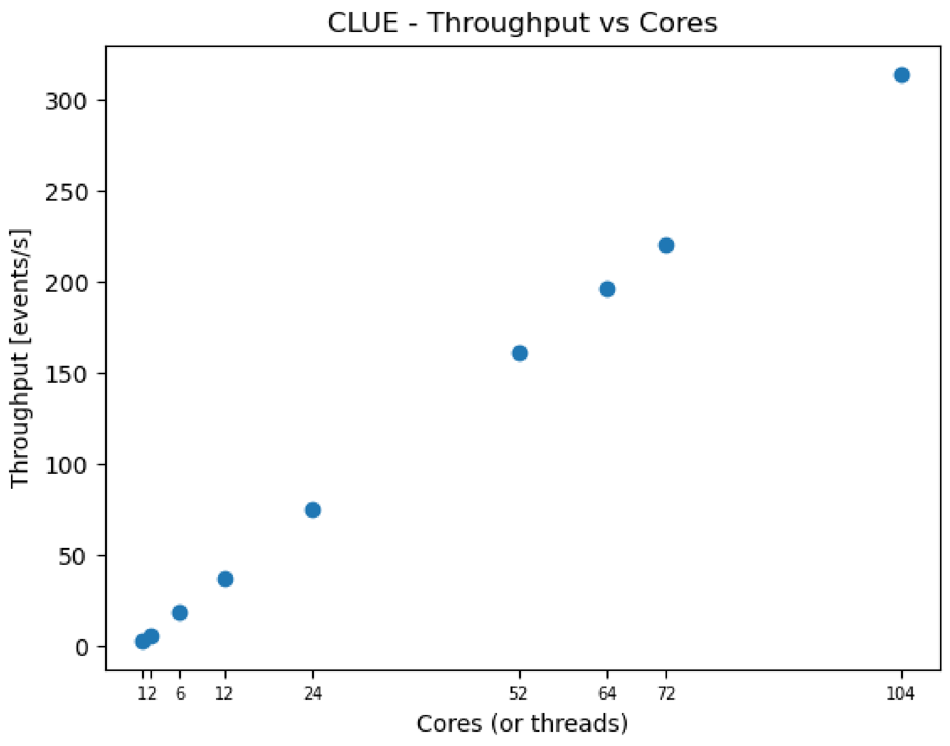


Figure 28: [HEP] Throughput vs number of used cores for CLUE on IDV-A CPU

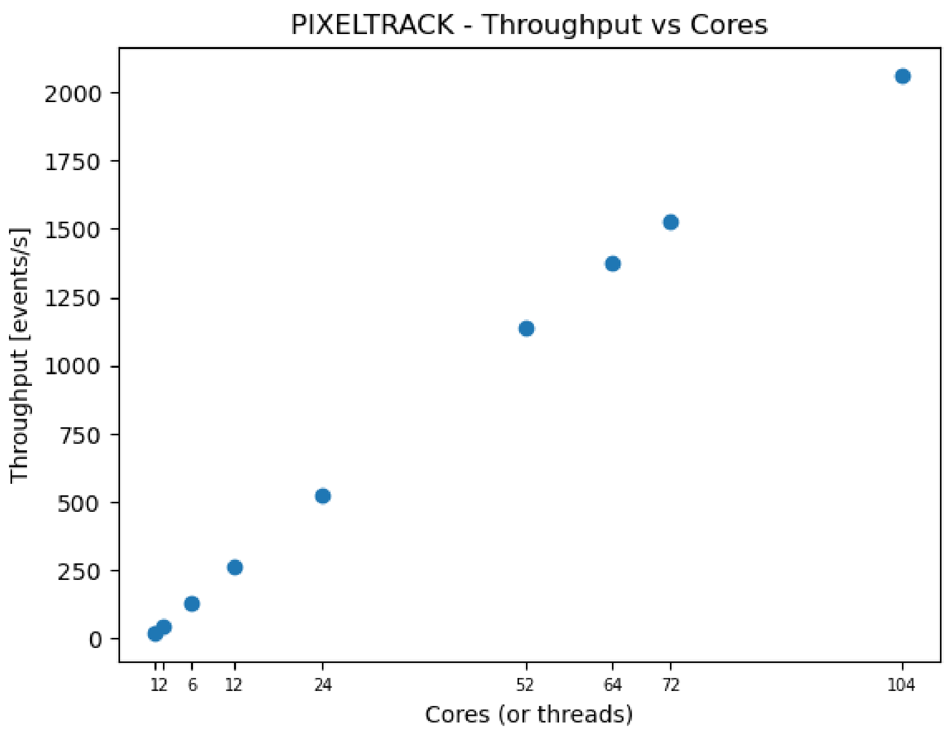


Figure 29: [HEP] Throughput vs number of used cores for Pixeltrack on IDV-A CPU

The scaling of the energy efficiency with the number of used CPU cores for the CLUE application on the IDV-E CPU (Figure 30) shows a similar linear trend to that on the Dibona CPU (Figure 22). Instead, for what regards the scaling on the energy efficiency on the IDV-A CPU of the Pixeltrack application, Figure 31shows a quite relevant improvement of the IDV-A CPU with respect to the corresponding result for the Dibona node (Figure 23). Focusing on the absolutes of the energy efficiency KPIs shows a significant degradation of performance on IDV-A tough. This is mainly due to the ~400 W power consumption in idle state of the node CPUs (with a ~200 W contribution accountable to the thermal control daemon in its current version).

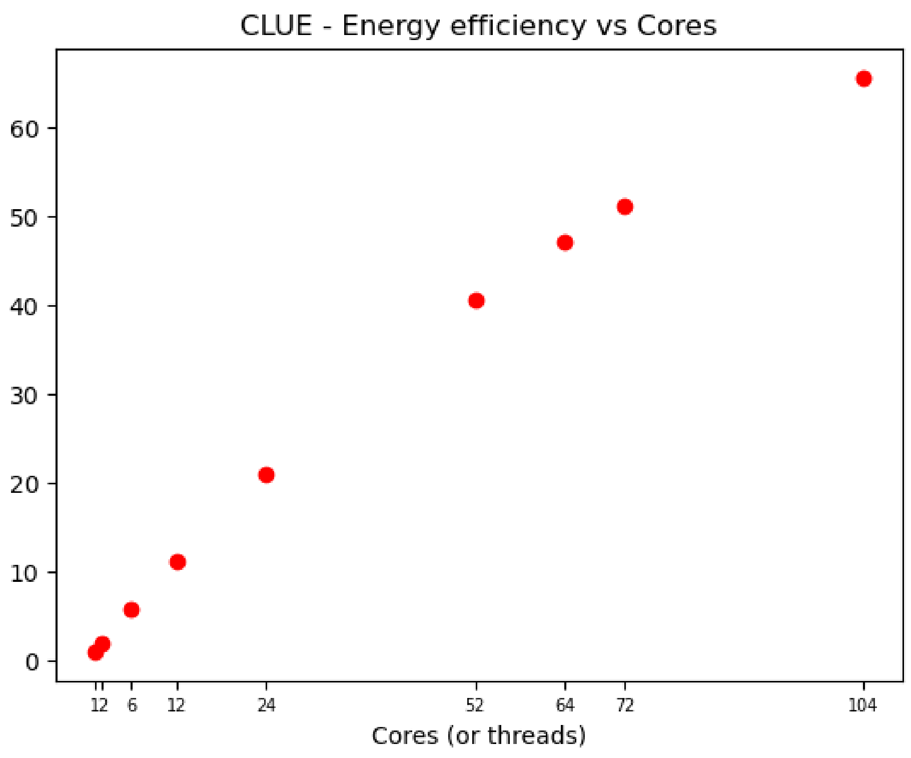


Figure 30: [HEP] Energy efficiency vs number of used cores for CLUE on IDV-A CPU

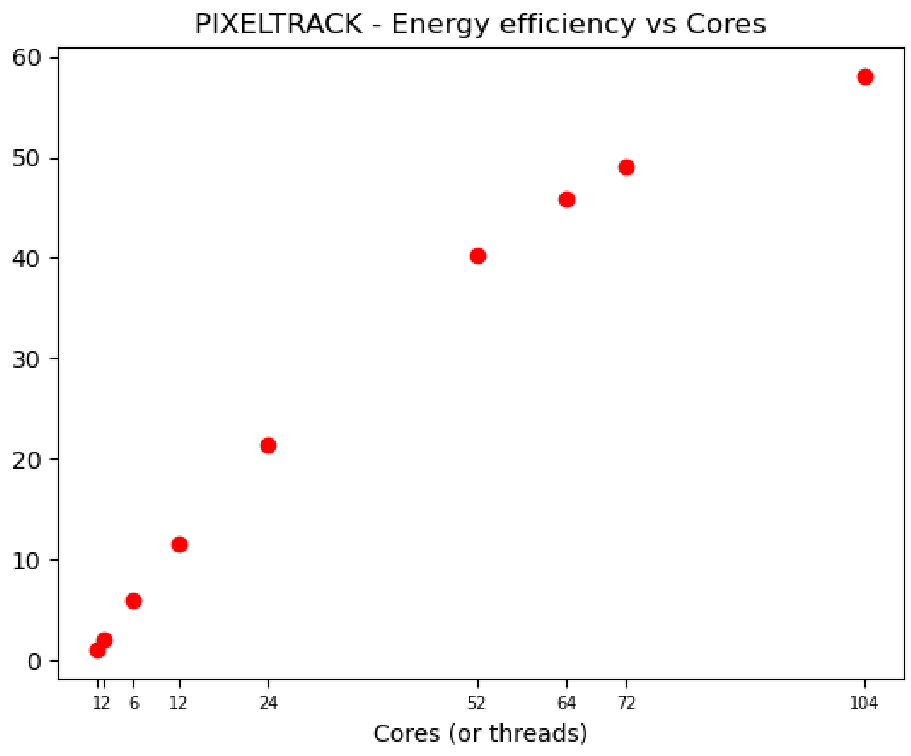


Figure 31: [HEP] Energy efficiency vs number of used cores for Pixeltrack on IDV-A CPU

2.4.2.2 Tests on GPU

We run the CLUE and Pixeltrack applications on the IDV-A node in the same configurations used to measure the baseline KPIs on the Dibona node and described in Sec.2.4.1.2 .
Measured KPIs scaling the number of used GPUs are reported below.

CLUE on GPU - H100			
GPU	Throughput (events/sec)	Power (W)	Energy efficiency (events/J)
1	1102.1	181.40	6,075
2	2194.63	364.50	6,021
3	3314.11	552.47	5,999
4	4300.63	728.38	5.90

Table 14: [HEP] KPIs for CLUE on GPU on IDV-A node

Pixeltrack on GPU - H100			
GPU	Throughput (events/sec)	Power (W)	Energy efficiency (events/J)
1	2507.31	182.678	13,725
2	4999.05	367.347	13,608
3	7212.48	546.521	13,197
4	9514.89	723.583	13,150

Table 15: [HEP] KPIs for Pixeltrack on GPU on IDV-A node

Moving from Dibona node (Table 11) the improvement in Pixeltrack throughput KPI is noticeable and expected in the IDV-A node (Table 15). The slightly worse performance of CLUE on IDV-A on throughput KPI (Table 14) compared to that measured on Dibona node (Table 10) needs further investigation, and possibly specific code tuning for the H100 GPU.

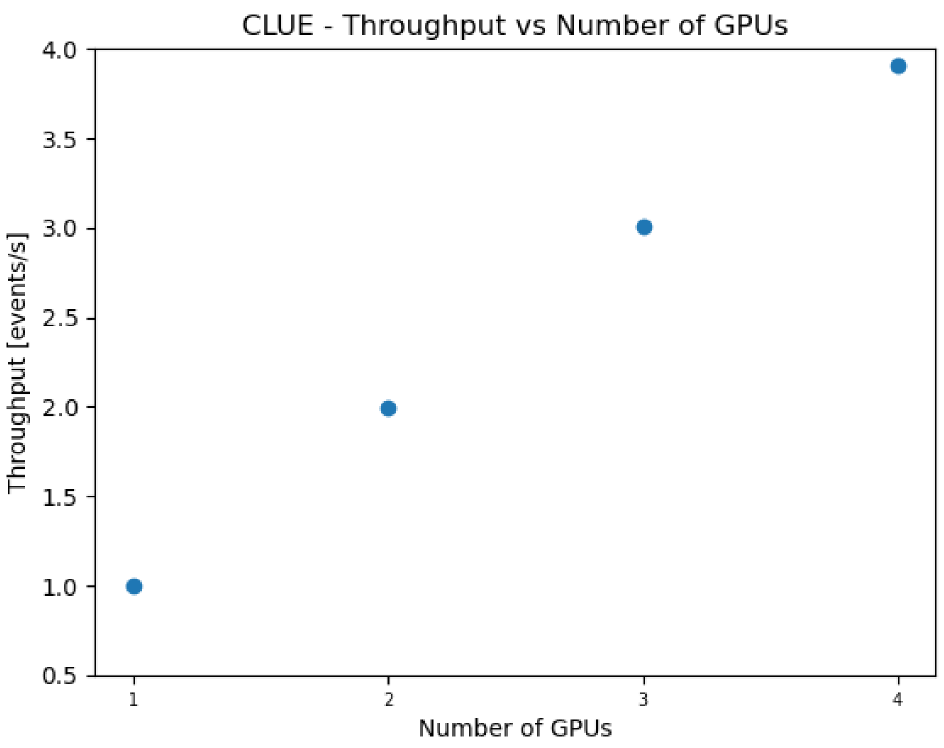


Figure 32: [HEP] Throughput vs number of used GPUs for CLUE on IDV-A node

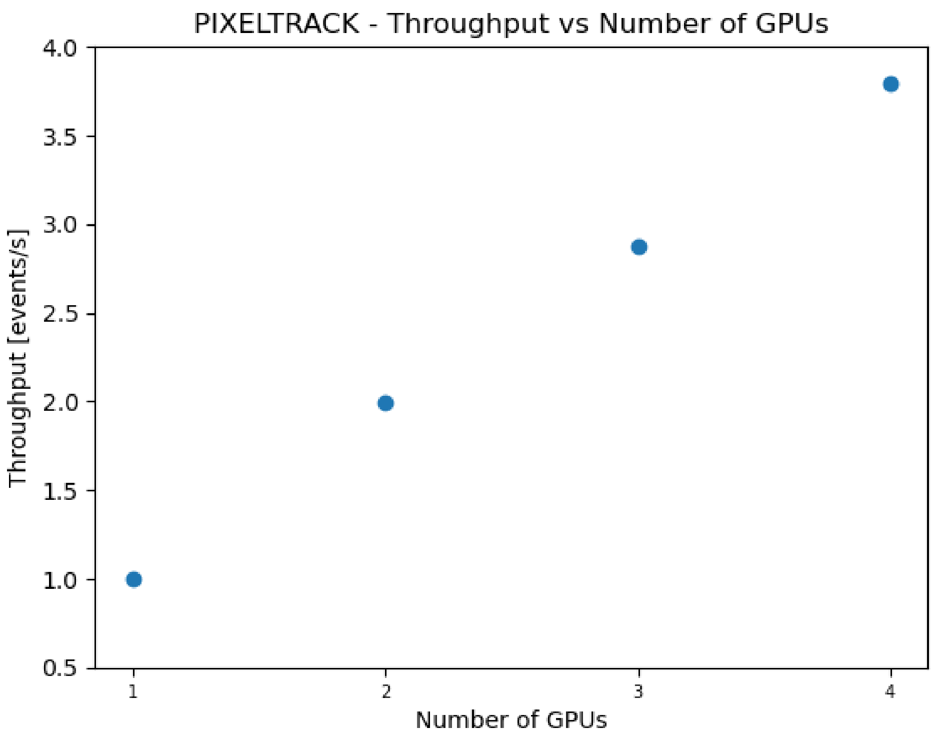


Figure 33: [HEP] Throughput vs number of used GPUs for Pixeltrack on IDV-A node

Figure 34 and Figure 35, on the other hand, show for both applications a slightly decreasing trend of the energy efficiency KPI (normalized to what is obtained on one GPU) scaling the number of the used GPUs on the IDV-A node.

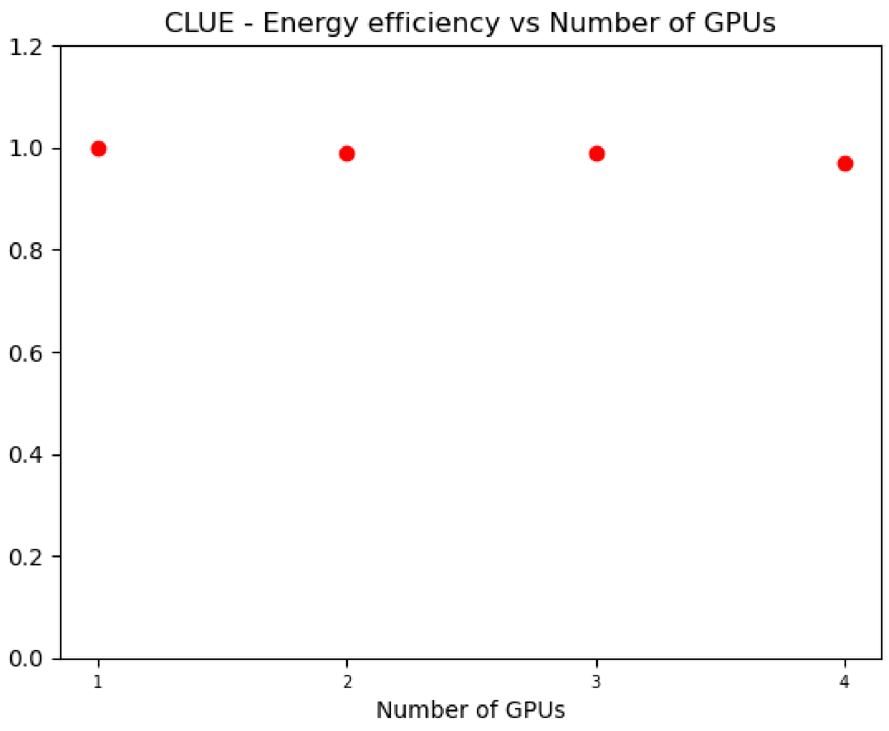


Figure 34: [HEP] Energy efficiency vs number of used GPUs for CLUE on IDV-A node

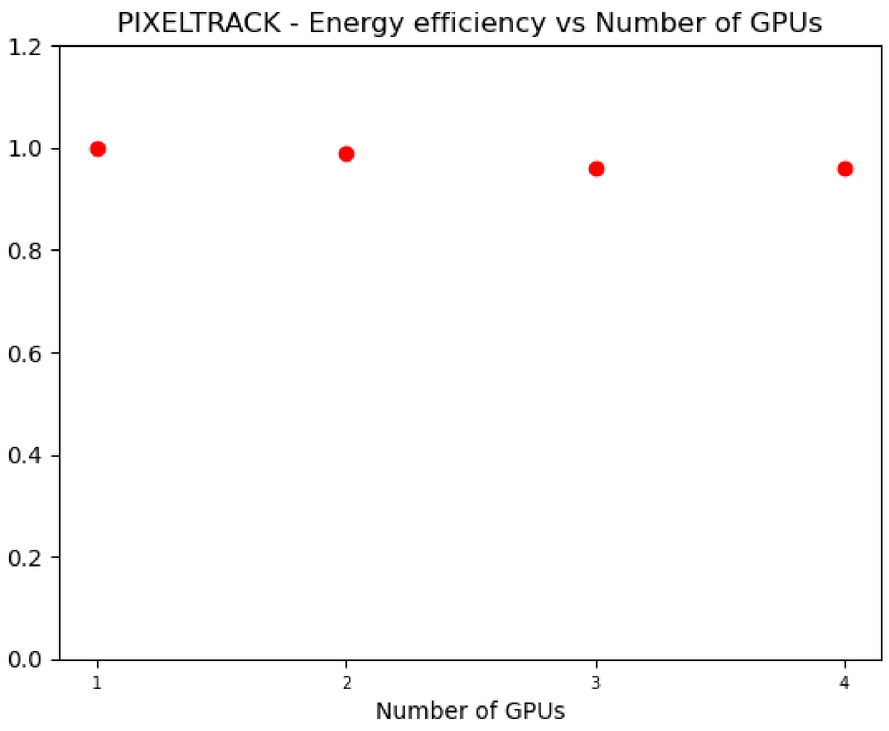


Figure 35: [HEP] Energy efficiency vs number of used GPUs for Pixeltrack on IDV-A node

2.4.3 Tests on IDV-E

Having the opportunity to run the benchmarks on the IDV-E nodes equipped with ARM64 Ampera Altra CPUs, we can report the performances of the aforementioned Pixeltrack on this server.

Pixeltrack on ARM64				
cores	**thread**	**Throughput (events/sec)**	**Power (W)**	**Energy efficiency (ref single core)**
1	1	24.863	64.80	1.0
2	2	58.582	63.38	2.409
4	4	112.156	71.35	4.097
8	8	212.776	74.65	7.429
16	16	410.435	89.20	11.992
32	32	778.272	109.64	18.500
64	64	1339.22	149.24	23.387
96	92	1691.69	166.86	26.423
128	128	1842.98	145.75	32.955
160	160	2011.04	212.74	24.637
192	192	2200.17	223.35	25.673
224	224	2017.87	208.94	25.170
256	256	2016.09	220.33	23.848

Table 16: [HEP] KPIs for Pixeltrack on CPU on IDV-E node

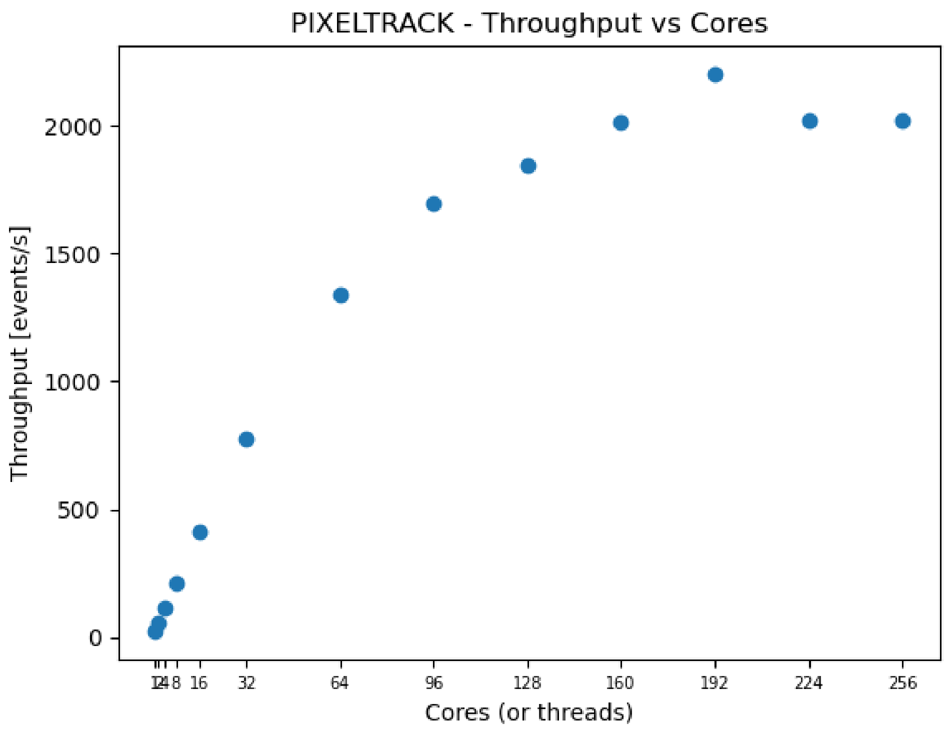


Figure 36: [HEP] Throughput vs number of used cores for Pixeltrack on IDV-E CPU

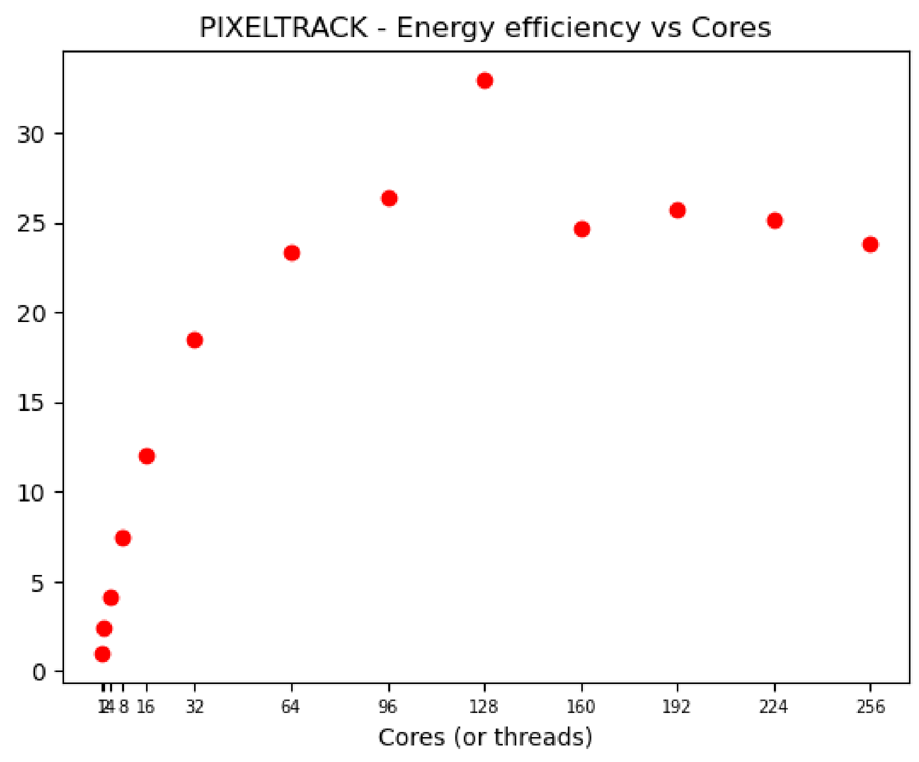


Figure 37: [HEP] Energy efficiency vs cores for Pixeltrack on IDV-E CPU

The IDV-E ARM64 architecture reaches the same throughput of the more performant of the two X86_64 ones (IDV-A) while surpassing by roughly a 50% the more energy efficient of them (Dibona) in the energy efficiency KPI.

2.5 NEST-GPU – INFN

As regards performance benchmarking in neuronal network simulations we mention the ongoing effort undertaken by the NEST team in the way of a standardization process [8] that designed a conceptual, generic workflow and produced a reference implementation, the beNNch framework. This is a set of modules for configuration, execution and analysis of benchmarks for NN simulations recording data and metadata in a unified way to foster reproducibility. Among the different modules we pick the test called *hpc_benchmark* [9]: it is a weak scaling benchmark employing a two populations network of excitatory and inhibitory leaky integrate-and-fire neurons, each with a fixed number of randomly drawn incoming connections (independent of the network size). This already has a multi-process implementation in the NEST repository as a Python script and we use an adapted version to perform the same test using the NEST-GPU application as a driver for the GPU as simulation engine. We mention in passing that the NEST-GPU has also received a new, GPU-accelerated implementation of the setup phase where the data structures pertaining to connections among neurons are created directly in GPU memory – a phase which at the time of D6.2 was still performed on CPU only and was quite time-consuming – that is described and benchmarked in detail in [10].

The power measurements were performed on the IDV-A platform using the GPowerU tool to encapsulate the launch of the *hpc_benchmark* script when run on the NVIDIA H100 GPUs (the same was done on the A100 GPUs equipped in the TEXTAROSSA partition in Dibona cluster for reference) and on the IDV-E platform by reporting at the same time the output of the *sensors* command – that returns the instantaneous power per CPU socket – together with the output of a Power Distribution Unit connected to the IDV-E that returns the whole power draw.

An interesting artifact of sampling the power output of an idle GPU with GPowerU is shown in Figure 38; here it can be clearly seen that the wattage sensor output is in discrete steps between 0.061W and 0.064W.and that idle power draw of a GPU hovers on a value over around 67W.

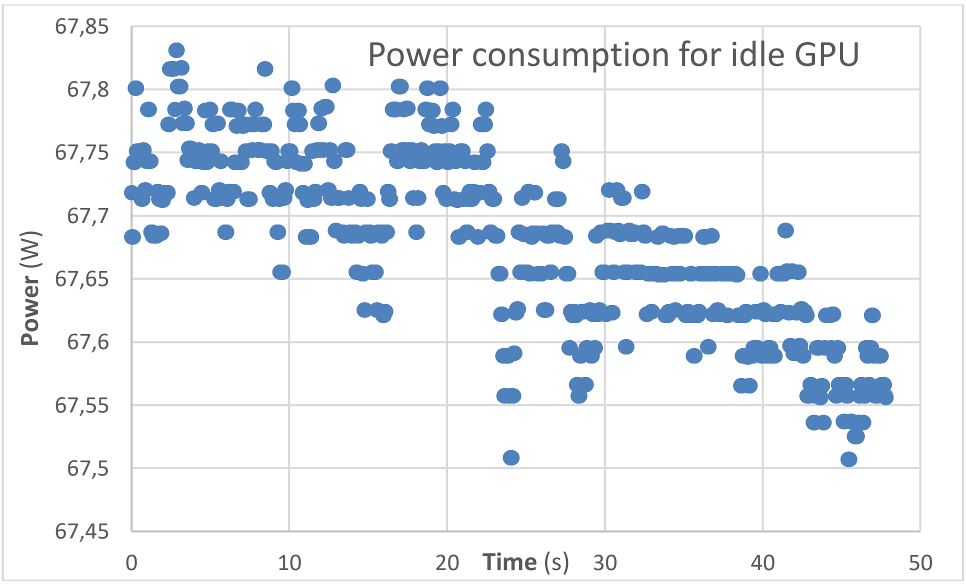


Figure 38: [NEST-GPU] Example of idle GPU power envelope

The test is configured to set up the network and perform a 'warm-up' 50ms pre-simulation in order to put the system into a steady state, followed by a 10.000ms simulation which is then timed; the results are in the following plots.

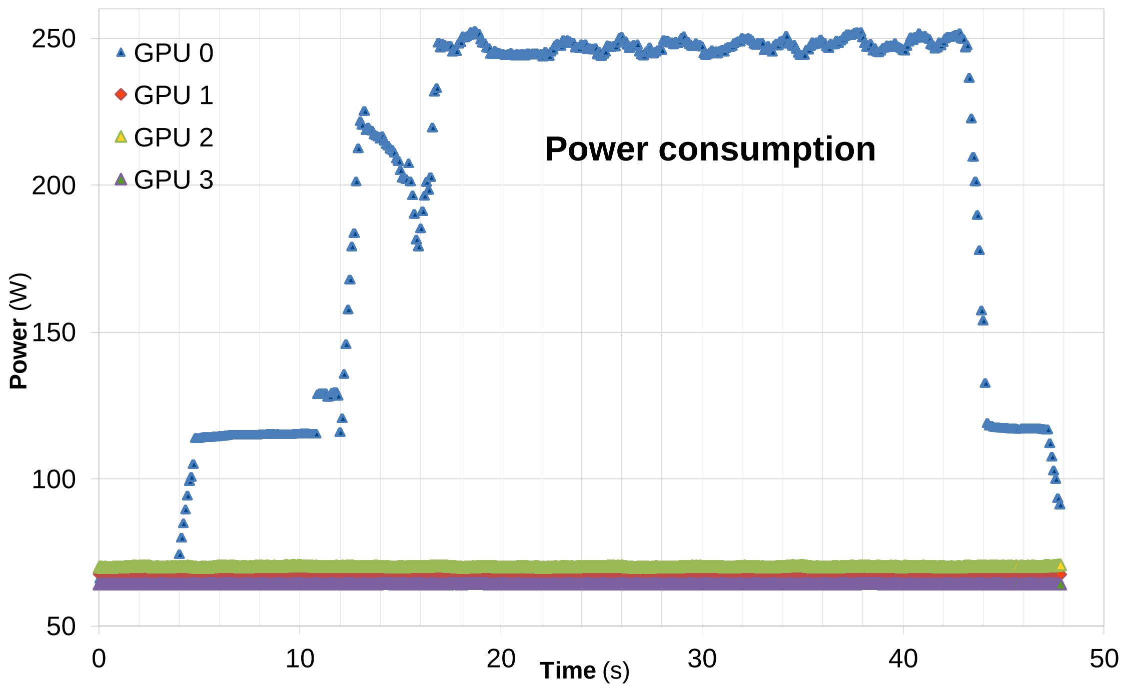


Figure 39: [NEST-GPU] Single active GPU power reading

From the plot in Figure 39 we can see that there is a network building phase – the first plateau – that pulls the power draw up from the idle state, then a 'pop' for the very first actual simulated milliseconds – 50ms which are required to put the network into a working 'steady state' and usually discarded in regular data taking –, a 'drop' when this is finished and then a more or less smooth line – the second plateau – for the remaining 10.000 simulated milliseconds.

When in this regime the average power draw that NEST-GPU can push a single H100 to reach is around **247W** with an absolute maximum of **252W**. If we take into consideration all GPUs the total draw is **449.5W**. Runtimes are **3.8s** for building and **27.3** for the actual simulation.

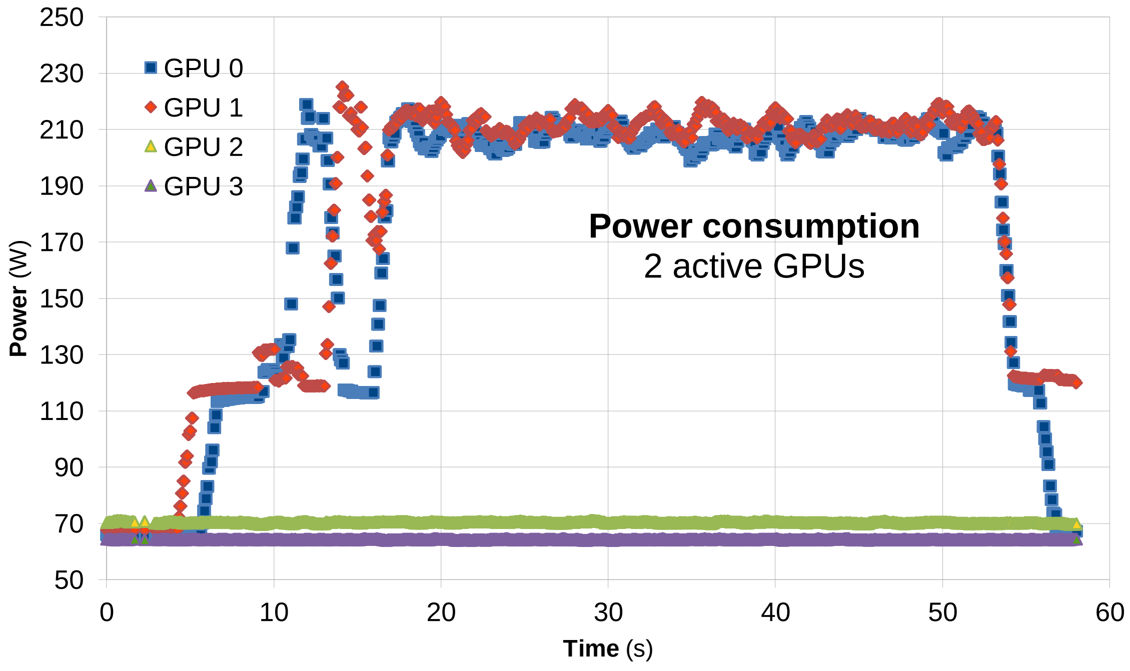


Figure 40: [NEST-GPU] 2 active GPUs power reading

As said, current testing is with a weak scaling benchmark; the same run is performed replicating a network of the same size on a second GPU and randomly connecting a fixed fraction of the neurons between the two. Here the simulation steps are necessarily interleaved with the exchange of messages between the GPU memories, which seems the reason why the throttling of the GPU frequencies by the thermal controller daemon is able to keep the average power draw in the steady state between **208W** and **212W** with and absolute maximum of **219W** and a grand total of **554W** including the idle ones. Here the building time is between **3.5s** and **4.4s** and the actual simulation takes **37.4s** on one GPU and **38.2s** on the other. It is apparent that adding inter-GPU communications – which are absent in the single GPU run – amounts to a significant 37% more runtime. It is being investigated how much this can be improved by optimizing the MPI calls – that currently require staging from the GPU memory to the CPU and back – with the GPU-aware ones that include support for direct accessing the GPU memory.

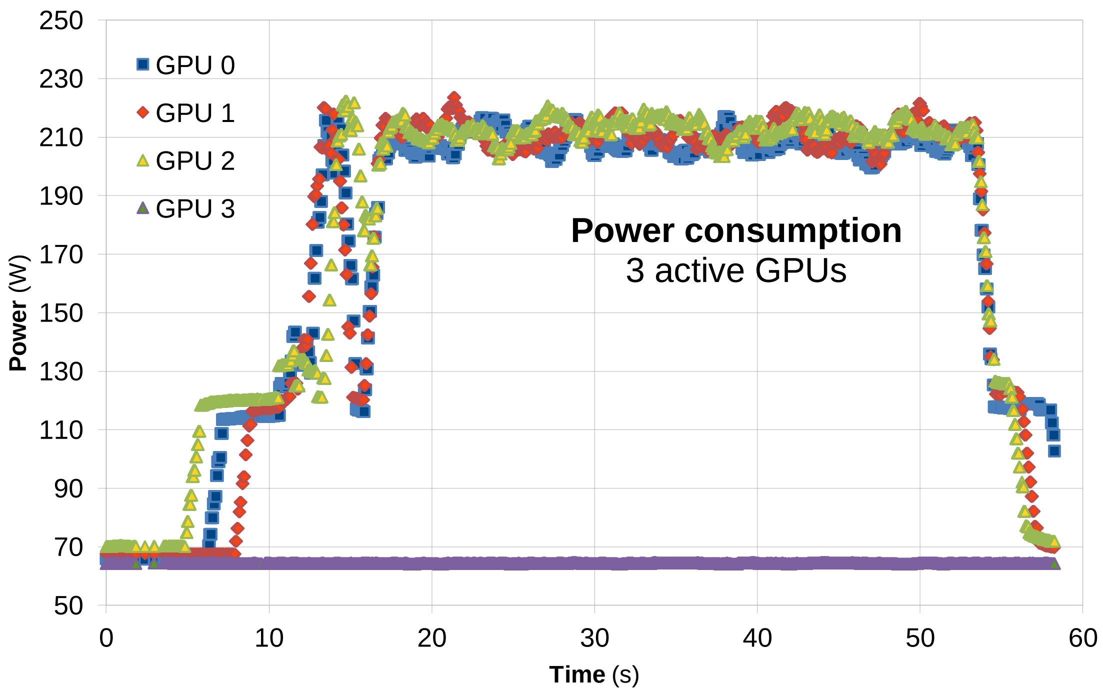


Figure 41: [NEST-GPU] 3 active GPUs power reading

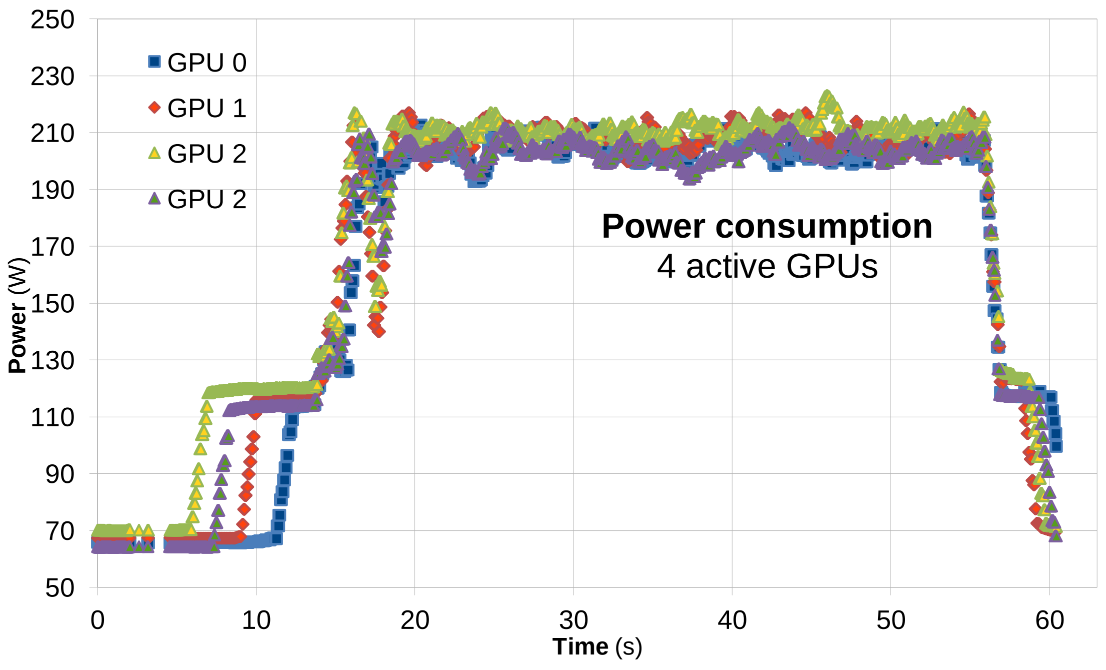


Figure 42: [NEST-GPU] 4 active GPUs power reading

The situation is similar for the 3 GPUs and the 4 GPUs cases; having accounted for the advantage of the no-comms situation for the single GPU, we see that the real weak scaling is quite good: building times are still around **3s** or less either for the 3 and 4 GPUs cases while actual simulation runtimes are between **37.7s** and **38s** for the 3 GPUs case and **38.4s** and **38.6s** for the 4 GPUs case, meaning a less than 3% difference in runtime between datasets which are two, three and four times the size of the single GPU case. Of course, the power consumption goes up as well, with 3 GPUs drawing between **208W** and **213W** on average in the steady state with a grand total of **697W** including the idle one and 4 GPUs drawing between **204W** and **211W** on average in the steady state and a grand total of **829W**.

For reference, we found that the very same problem run on the 4 NVIDIA A100 GPUs of the TextaRossa partition of the Dibona cluster runs consistently slower in the 2 and 4 GPUs cases – between **39s** and **40s** in the former and **42s** and **43s** in the latter – with a power draw that does not exceed **156W** per active GPU in the steady state – while the single GPU case is actually a little faster, with **24.7s** and a **184W** average power draw in the steady state.

The very same benchmark was performed with the CPU-only version of the NEST code running on the ARM cores on the IDV-E platform. Following the investigation done in D6.2 we directly chose the optimal process/cores layout for the platform, 16 MPI processes using 8 OpenMP threads each for a grand total of 128 busy cores and fixed the size to be the same of the single GPU run on the IDV-A. In the plot in Figure 43 is the polling of the internal probe for power consumption of both CPU sockets as returned by the *sensors* command with a 0.7s time interval between readings.

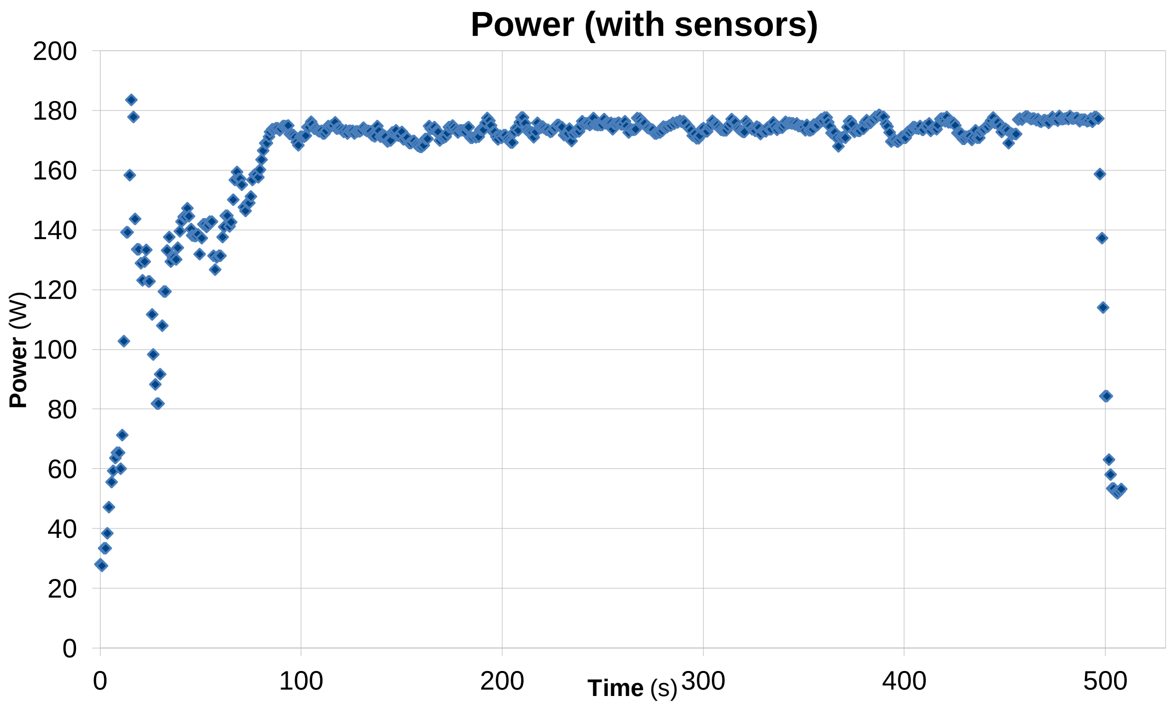


Figure 43: [NEST-GPU] IDV-E power reading

Here the building and pre-simulation phase are less distinguishable, appearing as an irregular slope up to the steady state plateau; maximum power draw is **184W** while average power in the steady state is **174W**, with **466s** for the 10.000 simulated milliseconds of network activity. For completeness, we report in Figure 44 the output of the Power Distribution Unit connected to the IDV-E node as queried by the system scripts available on the system. The slope is consistent with the previous plot, reaching a steady state after a little less that 100s; the total power output is reported as **416W** in the steady state (with a **467W** spike) which with an estimated 206W in idle (the lowest point in the plot) is consistent with more or less 200W more when running at full load.

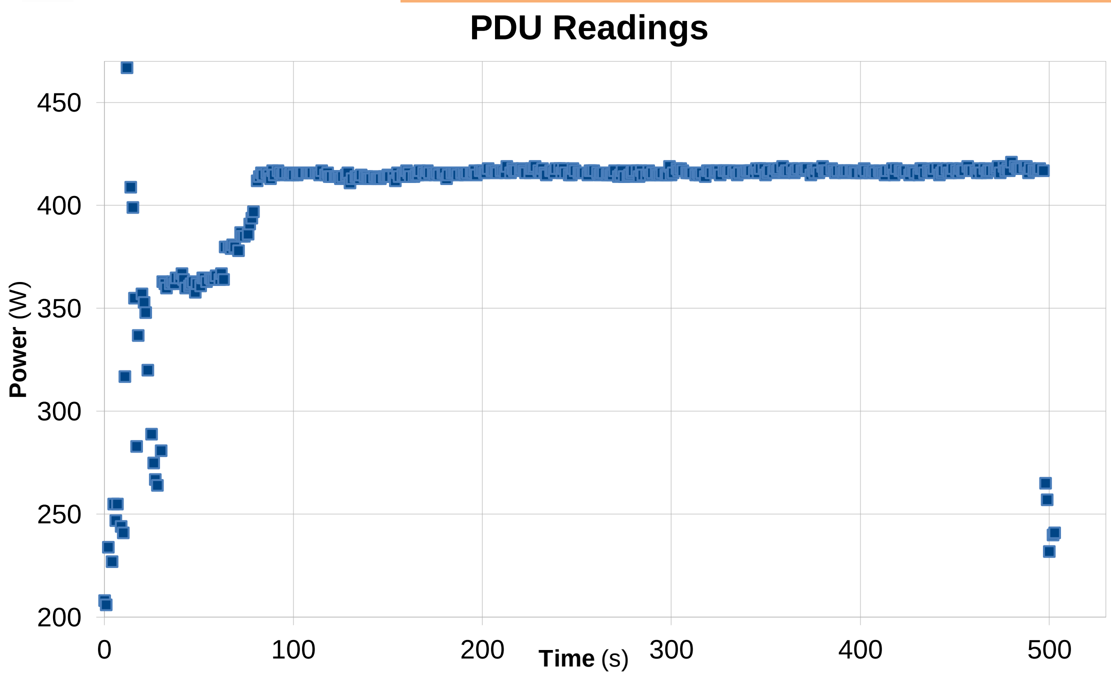


Figure 44: [NEST-GPU] IDV-E PDU reading

KPI	IDV-A single GPU	IDV-A multi-GPU	IDV-E
Simulated milliseconds/s	**366.3**	**259.1 ÷ 267.4**	**21.5**
Energy to solution (kJ)	**6.7**	**7.6 ÷ 8.2** (per GPU)	**81.1**
Synaptic updates/J	**1814.7**	**1600.0 ÷ 1482.8**	**149.9**

Table 17: [NEST-GPU] KPIs for the neural network application

2.6 RAIDER – INFN

In this section we report the results obtained from the execution of the RAIDER application implemented with APEIRON framework on the twin-FPGA IDV-E node first, and then on a couple of IDV-E nodes (four interconnected Xilinx U280 FPGA boards in total).
RAIDER is a high throughput online streaming processing application implemented on FPGA and its task is to perform particle identification (PID) on the stream of events generated by the RICH (Ring Imaging CHerenkov) detector in the CERN NA62 experiment, using neural networks.

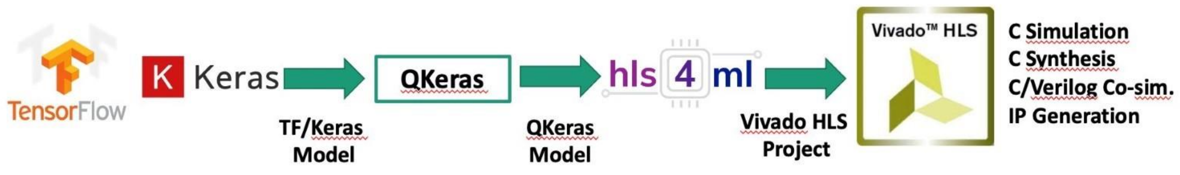


Figure 45: [RAIDER] The workflow for the generation of CNN kernels in RAIDER

In this case we adopt a Convolutional Neural Network (CNN) developed and trained offline by using Tensorflow/Keras, the first step of the workflow for the CNN generation shown in Figure 45.
The model receives as input a 16x16 image of the hit photomultipliers (PMTs) map - depicted in Figure 46 - for each physics event and its goal is to produce an estimate for the number of charged particles (0, 1, 2, >=3) for any RICH detector event, that corresponds to the number of ring tracks that can be reconstructed from the pattern of PMTs that have been illuminated by the Cherenkov light cone emitted by a charged particle traversing the detector.

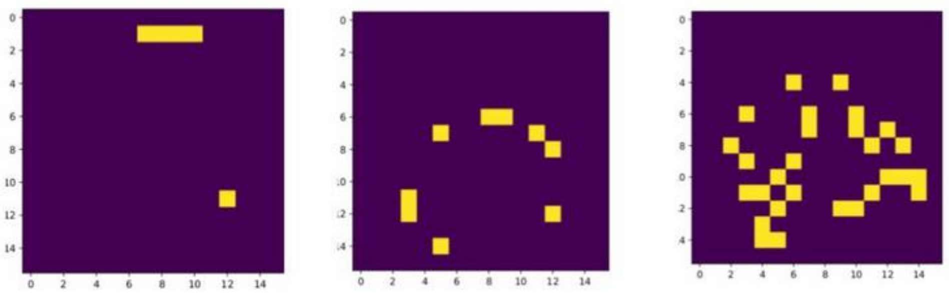


Figure 46: [RAIDER] Example of input images for the CNN

(left class 0, center class 1, right class 2).

To prepare the training and validation data for the CNN, we prepared different data sets composed by events extracted directly from NA62 database using the experiment analysis framework. The ground truth, used for training, was provided by the seedless offline reconstruction method.
To limit the FPGA resources footprint, as second stage, we performed a quantization step on the model using QKeras, resulting in two different fixed-point representations: <8, 1> for weights and biases and <16, 5> for activations.
As last step, the quantized model is translated into the corresponding Vitis HLS implementation using HLS4ML tool; implementation validated with Vivado C/Verilog co-simulation in terms of resources usage, performances (throughput and latency) and efficiency (referring to the classification accuracy of the CNN model). Lastly, the model can be synthetized as kernel IP to be integrated in the APEIRON framework and deployed on the FPGA.
Since the instantiation interval of the CNN obtained from HLS4ML scales with the size of the image, we expect to have a throughput for a single kernel implementation much smaller than the 10 MHz required by the NA62 L0 trigger. So, in order to improve the performances, RAIDER application has been designed with multiple processing kernels displaced in different nodes, capable to receive data/events from the network thanks the HAPECOM communication APIs.

We used two different setups to test the performance of this RAIDER implementation depicted in Figure 47 and Figure 56:

- Four interconnected Xilinx® Alveo U280 installed on the IDV-E node in a ring topology, 2 FPGAs for each of the two IDV-E nodes.
- Four interconnected Xilinx® Alveo U200 installed in the INFN Roma APE Lab in a ring topology, one FPGA for each of the four single Intel(R) Xeon(R) Silver 4410T CPU nodes.

In particular, we are interested in studying the throughput of the overall system, since it is one of the most important requirements for the integration in trigger and data acquisition systems in HEP experiments. The interconnected boards are used as nodes of a RAIDER deployment via APEIRON framework with distinct roles:

- **Preprocessing node**: data are loaded from Host memory and sent through the network via an HLS kernel (*krnl_sender*). Data are then processed by 3 Imagifier HLS kernels which turn the PMT hitlist information into a 256bit word (16x16 B&W image) that is sent to the Computing node through the internode ports of the INFN Communication IP. As second task, this node is in charge of receiving the output of the CNN computation and storing it on Host memory via an HLS kernel (*krnl_receiver*). The processing time, from the first packet sent to the last received, is measured on this node.
- **Computing node**: images coming from external links are taken as input and dispatched to various CNN HLS kernels (each of them connected to a different INFN Communication IP intranode port) to compute the predictions. Results are then sent back to the preprocessing node.

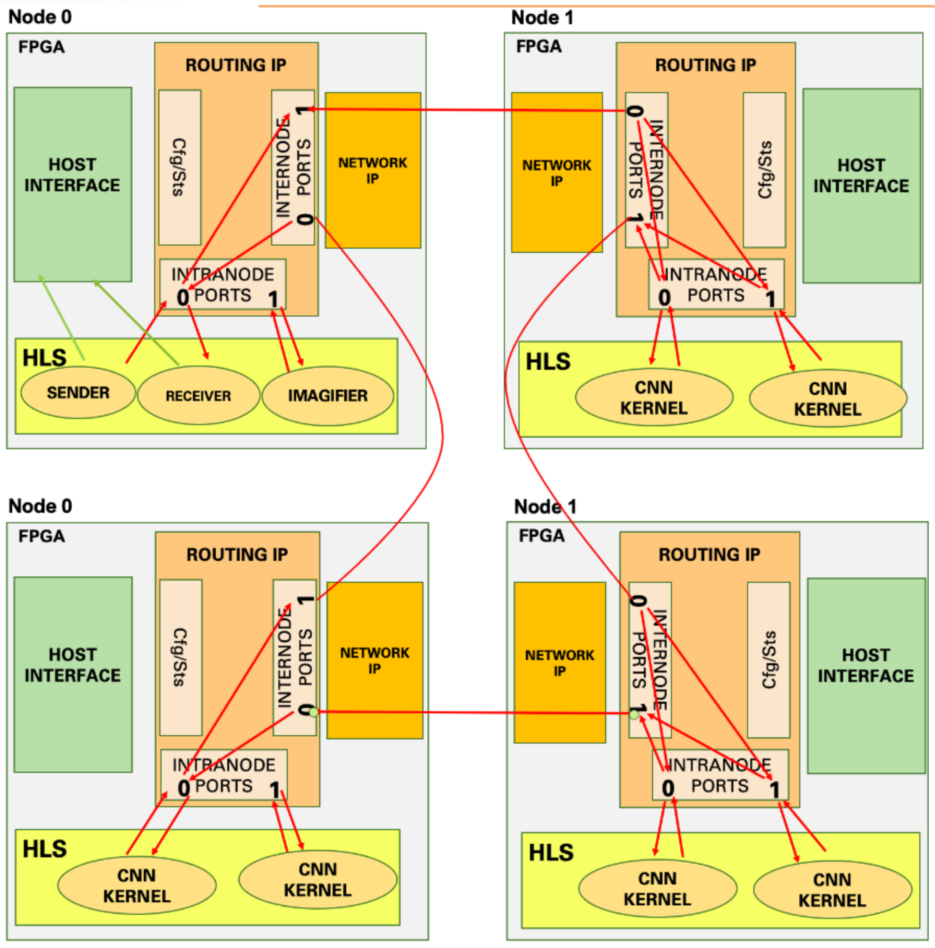
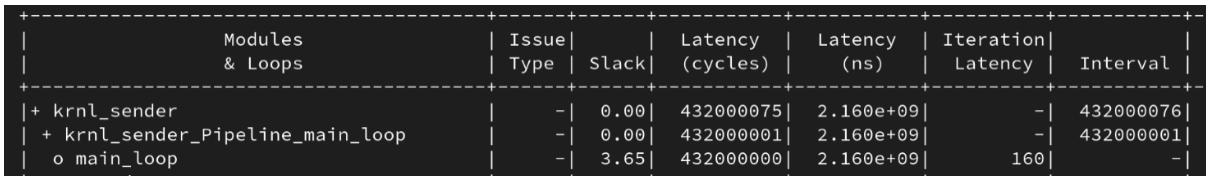


Figure 47: [RAIDER] Test setup on the Xilinx® Alveo U200 installed in the INFN Roma1 APE Lab

All tests performed in both testbeds have been done by using a 200 MHz global clock in the hardware setup. This clock increasing with respect to the one used for the D6.2 measures has been possible thanks to the improvement of the INFN Communication IP (described in detail in the D2.9). Another difference with respect to the D6.2 application is that the data - to be sent through the network from the preprocessing node – are now loaded from the BRAM instead of the DDR FPGA memory. This change is due to a limit that we've seen while testing the application loading events from DDR memory: while increasing the number of CNN kernels in the setup to test RAIDER scaling, we noticed that, working with 3 CNNs or more, the throughput reachable was always of 1.278MHz. This corresponds to the instantiation interval of the *sender* HLS kernel of ~160 clock cycles, as can be seen from the report obtained in output of the compiling Vitis process depicted in Figure 48.



```
+-----------------------------------------+-------+------+-----------+-----------+----------+-----------+
|                 Modules                 | Issue |      |  Latency  |  Latency  | Iteration|           |
|                 & Loops                 | Type  | Slack|  (cycles) |   (ns)    |  Latency |  Interval |
+-----------------------------------------+-------+------+-----------+-----------+----------+-----------+
|+ krnl_sender                            |      -|  0.00|  432000075| 2.160e+09|         -|  432000076|
| + krnl_sender_Pipeline_main_loop        |      -|  0.00|  432000001| 2.160e+09|         -|  432000001|
|  o main_loop                            |      -|  3.65|  432000000| 2.160e+09|       160|          -|
```

Figure 48: [RAIDER] Vitis synthesis report of krnl_sender HLS kernel

For this reason, we decide to work with data loaded from the BRAM for the whole tests reported in this deliverable (in both the testbeds).

2.6.1 RESULTS ON APE LAB TESTBED

In this setup, we have scaled the system starting from tests on 2 nodes (one preprocessing and one computing) up to 4 (adding 2 more computing nodes), changing from the host code (at computing kernels launch) the number of running CNN on each computing node.
For what concerns energy efficiency, CPU hosts power measurements have been performed using *turbostat:* a Linux command-line utility that reports processor topology, frequency, idle power-state statistics, temperature, and power on X86 processors. The measurement is performed concurrent to application execution on each node present in the used setup.

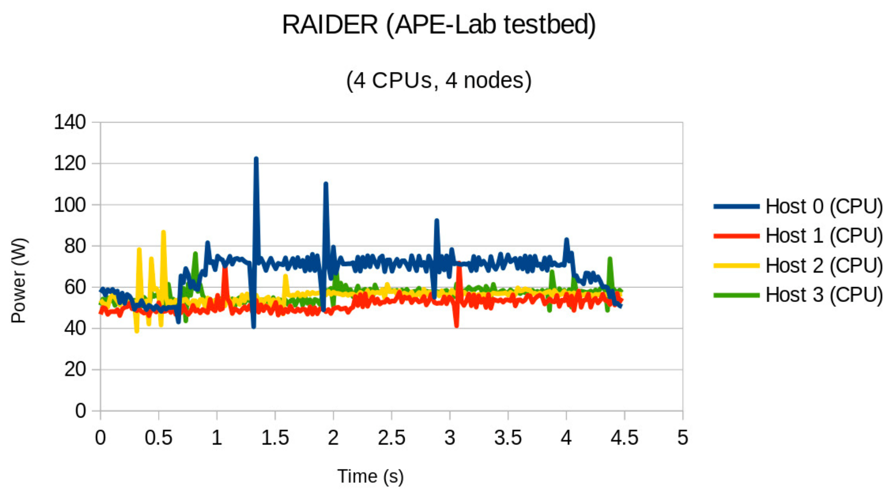


Figure 49: [RAIDER] Application CPU hosts power profiles (4 Intel Sapphire Rapids)

For tracking accelerator cards power profiles, FPGA setups power measurements, concurrent with the HLS kernels execution on both type of nodes, were extrapolated for the XRT summary .csv file obtained in output from the CPU host application. Following the node and number of

CNNs scaling, power profiles have been produced for each of the RAIDER configuration tested and are reported in Figure 50, Figure 51 and Figure 52.

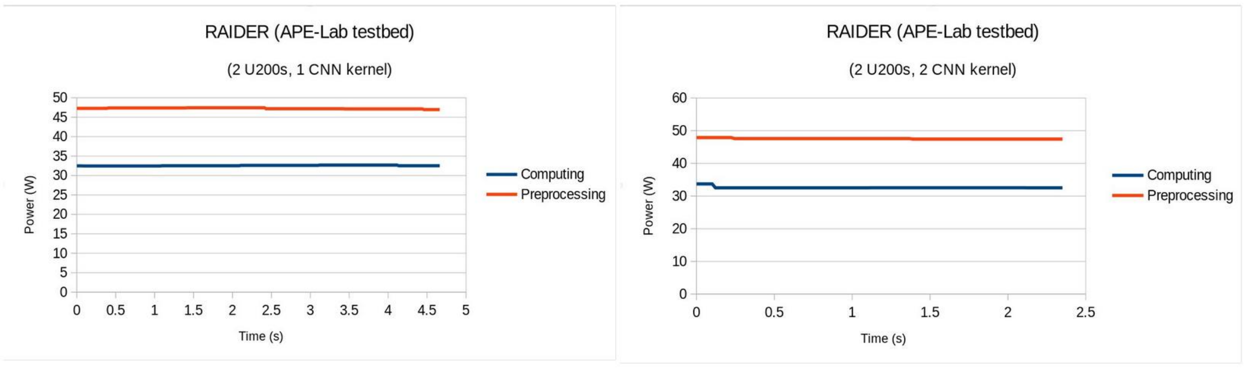


Figure 50: [RAIDER] FPGA power profiles (2 Alveo Xilinx U200 setup)

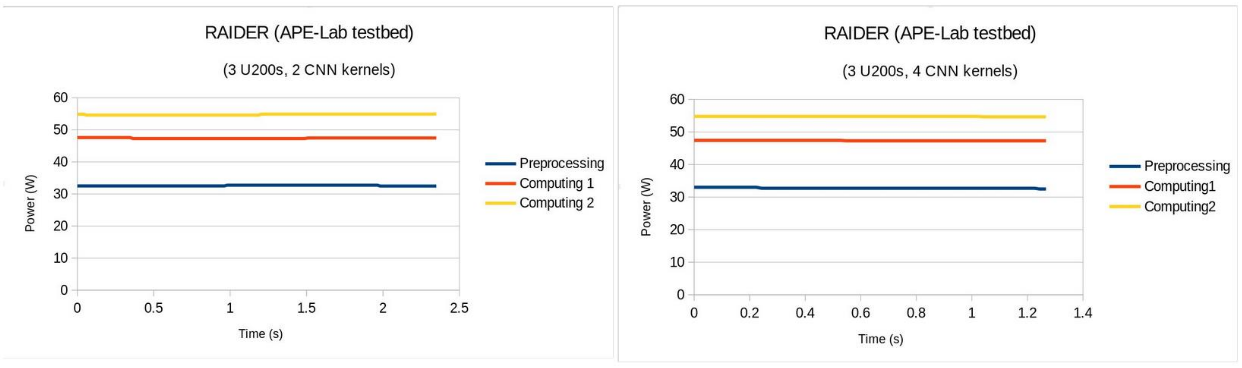


Figure 51: [RAIDER] FPGA power profiles (3 Alveo Xilinx U200 setup)

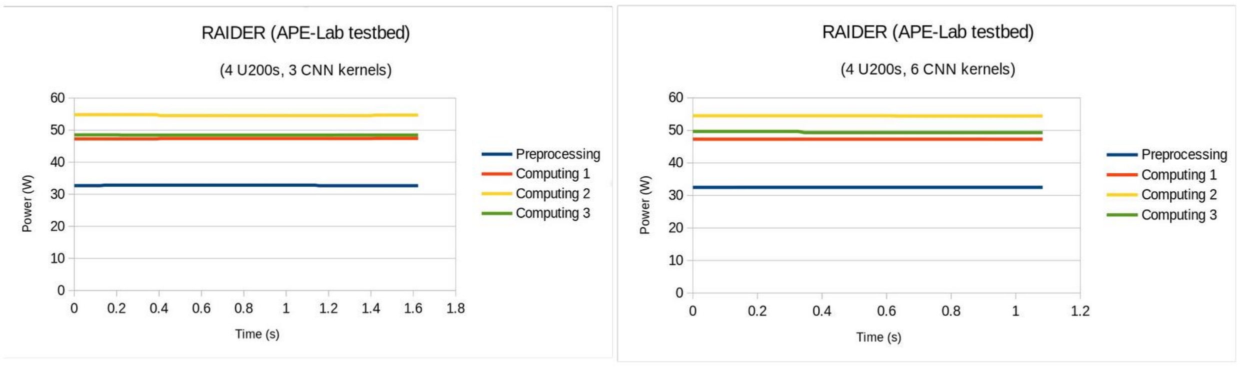


Figure 52: [RAIDER] FPGA power profiles (4 Alveo Xilinx U200 setup)

The integrated processing throughput of the RAIDER application deployed on the APE Lab testbed has been measured host-side by scaling the number of implemented computing nodes. Results are tabulated in Table 18 and plotted in Figure 53, together with the energy efficiency values obtained by integrating the power profiles measured for all the tested setups.

# computing nodes	# CNNs	Throughput (Mevents/s)	Energy efficiency (kevents/J)
1 node	1 CNN	0.581	3.015
	2 CNNs	1.163	6.005
2 nodes	2CNNs	1.163	3.838
	4CNNs	2.325	7.692
3 nodes	3CNNs	1.744	4.298
	6CNNs	2.692	6.626

Table 18: [RAIDER] Processing Throughput with an increasing number of Computing nodes (and CNN HLS kernels)

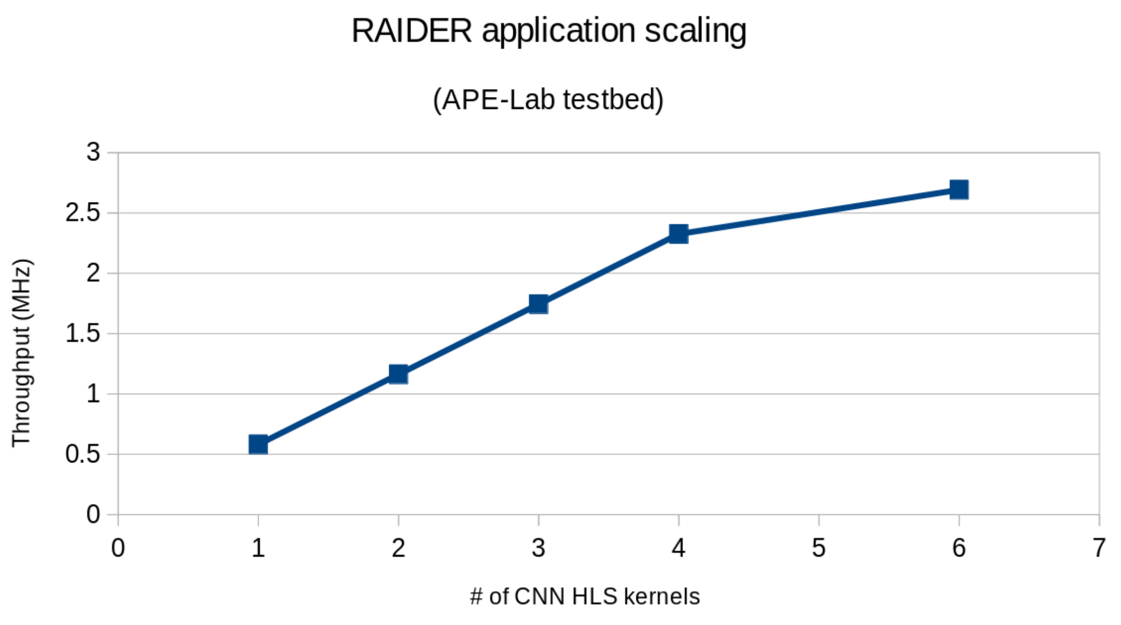
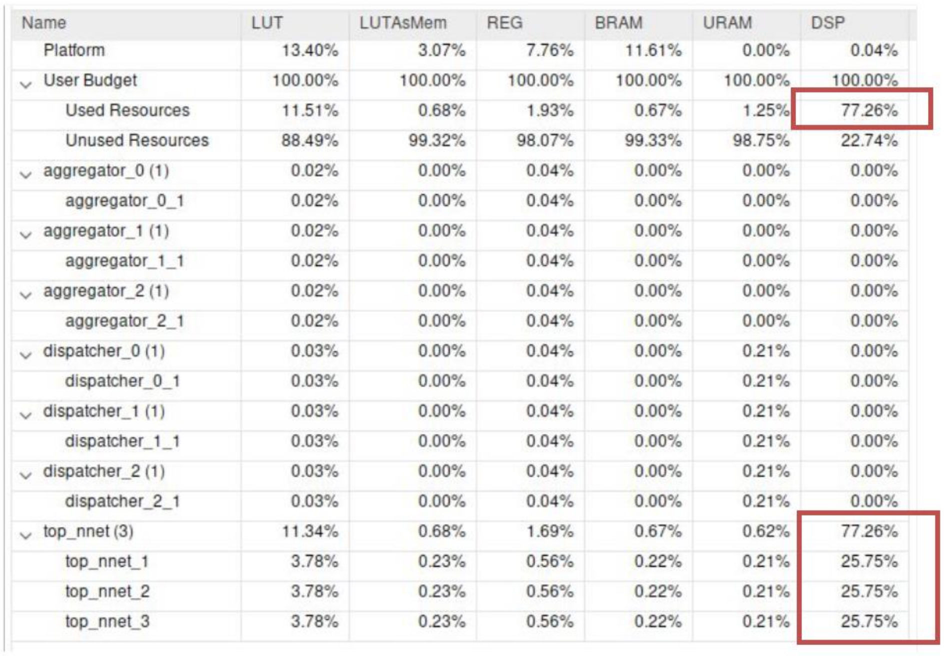


Figure 53: [RAIDER] Throughput scaling with an increasing number of CNN HLS kernels

The presented results show the good scaling of system performance with the number of nodes, while the flattening slope of the curve when the number of CNNs goes beyond 4 is mainly due to the saturation of the data injection rate in the *krnl_sender* and the instantiation interval of the *imagifier* (~70 clock cycles) in the preprocessing node.

2.6.2 U280 RESULTS ON IDV-E

In this setup, we have tested the system starting on 2 FPGAs (one preprocessing and one computing) up to 4 (adding 2 more computing FPGAs) with a slightly different setup with respect to the one used in the U200 APE Lab testbed. Since the resources available on the Xilinx Alveo U280 (for example, more DSP) are larger with respect to the U200 ones, we could implement 3 CNNs HLS kernels on single computing FPGA computing (a Vitis report of the hardware occupation is reported in Figure 54) since a single CNN takes ~25% of the maximum DSP resource.



Name	LUT	LUTAsMem	REG	BRAM	URAM	DSP
Platform	13.40%	3.07%	7.76%	11.61%	0.00%	0.04%
User Budget	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Used Resources	11.51%	0.68%	1.93%	0.67%	1.25%	77.26%
Unused Resources	88.49%	99.32%	98.07%	99.33%	98.75%	22.74%
aggregator_0 (1)	0.02%	0.00%	0.04%	0.00%	0.00%	0.00%
aggregator_0_1	0.02%	0.00%	0.04%	0.00%	0.00%	0.00%
aggregator_1 (1)	0.02%	0.00%	0.04%	0.00%	0.00%	0.00%
aggregator_1_1	0.02%	0.00%	0.04%	0.00%	0.00%	0.00%
aggregator_2 (1)	0.02%	0.00%	0.04%	0.00%	0.00%	0.00%
aggregator_2_1	0.02%	0.00%	0.04%	0.00%	0.00%	0.00%
dispatcher_0 (1)	0.03%	0.00%	0.04%	0.00%	0.21%	0.00%
dispatcher_0_1	0.03%	0.00%	0.04%	0.00%	0.21%	0.00%
dispatcher_1 (1)	0.03%	0.00%	0.04%	0.00%	0.21%	0.00%
dispatcher_1_1	0.03%	0.00%	0.04%	0.00%	0.21%	0.00%
dispatcher_2 (1)	0.03%	0.00%	0.04%	0.00%	0.21%	0.00%
dispatcher_2_1	0.03%	0.00%	0.04%	0.00%	0.21%	0.00%
top_nnet (3)	11.34%	0.68%	1.69%	0.67%	0.62%	77.26%
top_nnet_1	3.78%	0.23%	0.56%	0.22%	0.21%	25.75%
top_nnet_2	3.78%	0.23%	0.56%	0.22%	0.21%	25.75%
top_nnet_3	3.78%	0.23%	0.56%	0.22%	0.21%	25.75%

Figure 54: [RAIDER] Vitis synthesis report of hardware project to be deployed on Computing node.
In red, the DSP utilization is reported.

In addition to this, taking into account the limitations in terms of throughput found in the APE Lab testbed, we decided to deploy up to 3 *imagifier* HLS kernel on a single preprocessing FPGA. The resulting hardware setup is so reported in Figure 55 and Figure 56, in which are depicted, respectively, the design used for single and double node RAIDER execution on IDV-E testbed.

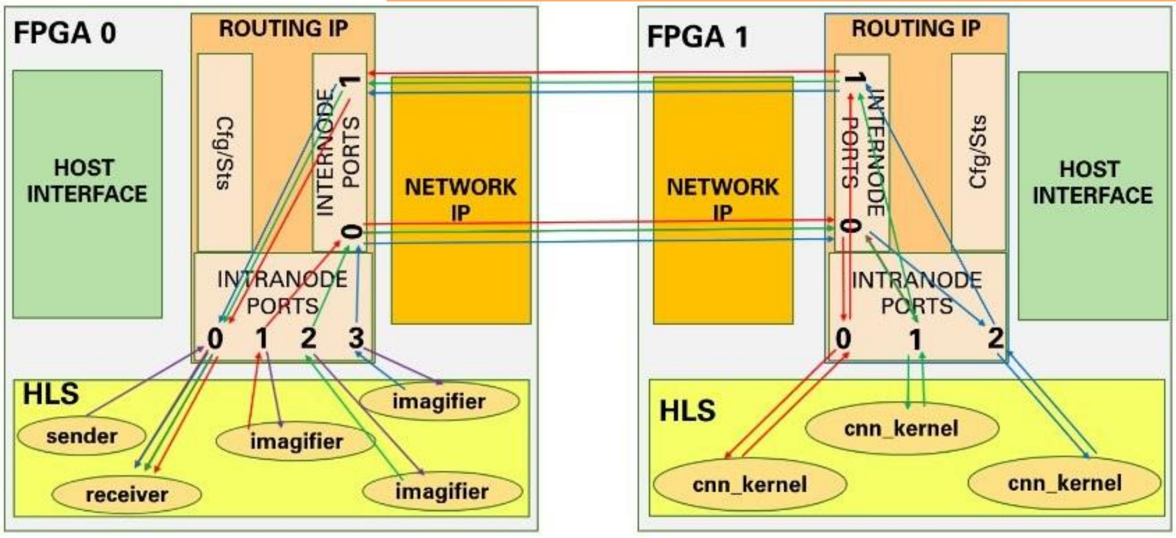


Figure 55: [RAIDER] Test setup on the Xilinx® Alveo U280 installed in the IDV-E testbed (single node)

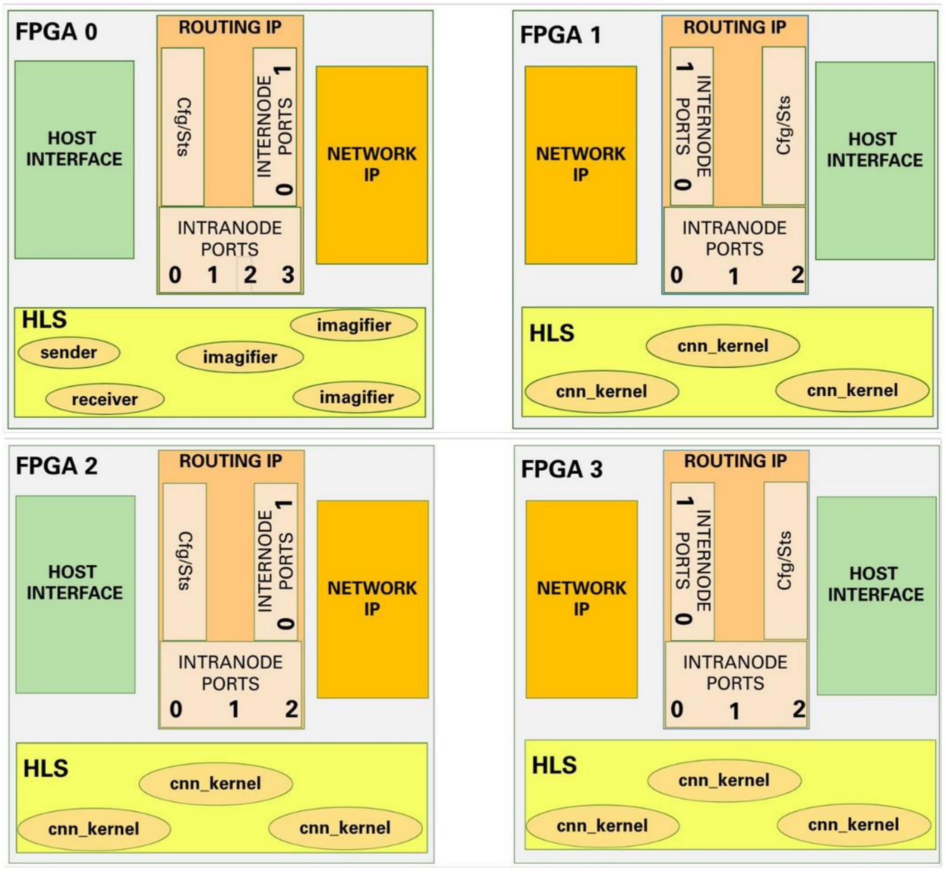


Figure 56: [RAIDER] Test setup on the Xilinx® Alveo U280 installed in the IDV-E testbed (double node)

The integrated processing throughput of the system has been measured host-side and results are tabulated in Table 19.

# computing FPGAs	# CNNs	Throughput (Mevents/s)
1 FPGA	3 CNNs	1.813
2 FPGAs	6 CNNs	3.409
3 FPGAs	9 CNNs	4.874

Table 19: [RAIDER] processing time per event with an increasing number of Computing FPGAs (and CNN HLS kernels)

For what concerns energy efficiency, CPU hosts power measurements have been performed by using *sensors:* a free and open-source application that provides tools and drivers for monitoring temperatures, voltage, and fans. The measurement has been performed concurrent to application execution on each node present in the used setup and are reported in Figure 57. From CPU power profiles, it is possible to notice a ~4W difference between the two nodes used in the IDV-E testbed.

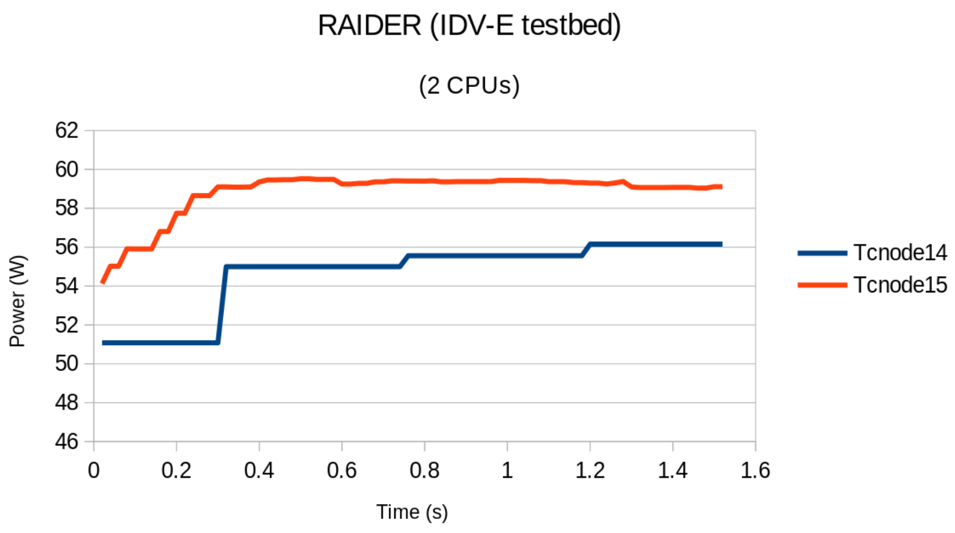


Figure 57: [RAIDER] CPU hosts power profiles (2 Ampere Altra Max processor)

For tracking accelerator cards power profiles, FPGA setups power measurements, as for the APE Lab testbed case, were extrapolated for the XRT summary .csv file obtained in output from the CPU host application. As can be seen from Figure 58, Figure 59 and Figure 60, Computing FPGAs 2 and 3 seems to have a power average slightly larger than the node 1: this is probably due to the fact that the nodes 2 and 3 are placed in a different IDV-E server with respect to the one in which the node 0 and 1 are In particular computing FPGA 2 and computing FPGA 3 are hosted in a air-cooled node (tcnode14), while computing FPGA 1 is hosted in node tcnode15 equipped with the two-phase cooling system developed in TEXTAROSSA.

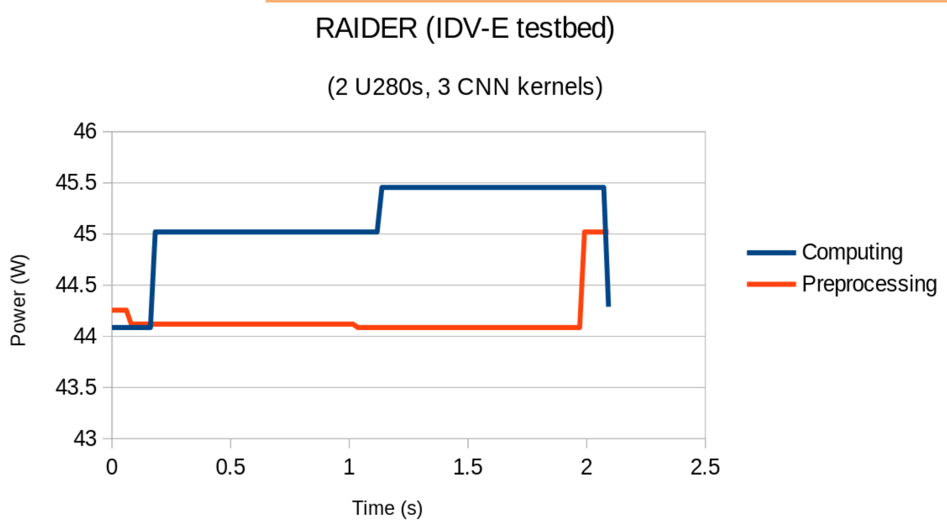


Figure 58: [RAIDER] FPGA power profiles (2 Alveo Xilinx U280 setup)

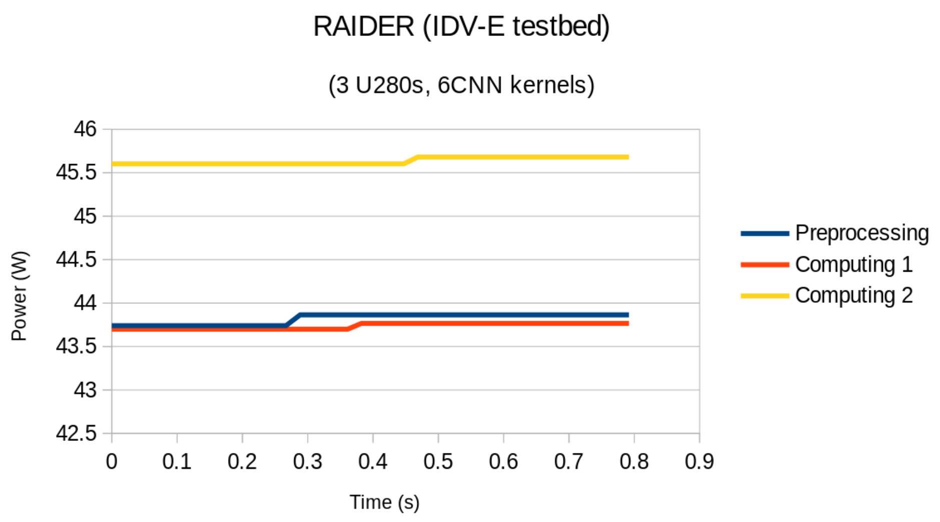


Figure 59: [RAIDER] FPGA power profiles (3 Alveo Xilinx U280 setup)

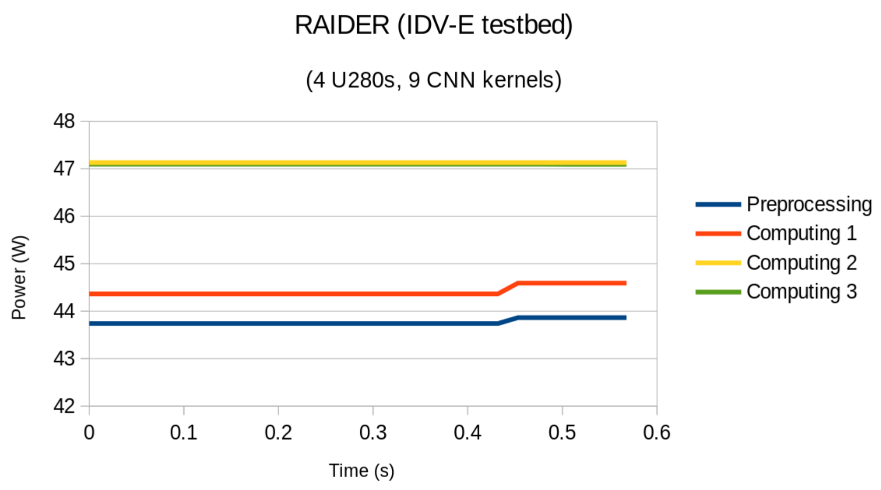


Figure 60: [RAIDER] FPGA power profiles (4 Alveo Xilinx U280 setup)

KPIs

To make a comparison with the KPIs presented in D6.2, we report as baseline the performance values obtained for the RAIDER setup in the previous deliverable in Table 20. For what concern the energy to solution and energy efficiency measurements, we considered correct to insert the relative FPGA energy consumption together with the node total energy data: this allows us to disentangle the different contributions to the total energy consumption and highlight the performance of the FPGA accelerator board alone.

Our goal is to show the updates of the application with respect to the past one and the scaling results obtained via the capabilities of the APEIRON framework and the adoption of the IDV-E platform.

KPI	RAIDER @100 MHZ [1 FPGA, 1CNN]	RAIDER @100 MHZ [1 FPGA, 2CNNs]
purity/efficiency (per class)	efficiency: - 0: 92% - 1: 79% - 2: 75% - 3+: 76% purity: - 0: 83% - 1: 88% - 2: 70% - 3+: 80%	efficiency: - 0: 92% - 1: 79% - 2: 75% - 3+: 76% purity: - 0: 83% - 1: 88% - 2: 70% - 3+: 80%
time to solution [s]	9.701	4.898
throughput [events/s]	**278324.152**	**551245.410**
energy to solution [J]	563.174 (262.090 FPGA)	267.831 (137.902 FPGA)
energy efficiency [events/J]	**4794.255** **(10301.805 FPGA)**	**10079.328** **(19579.121 FPGA)**

Table 20: [RAIDER] Baseline KPIs evaluated on the execution of the RAIDER application on a single Xilinx Alveo U200 FPGA, processing 2.7M events with a global clock of 100 MHz

Since the updates of the INFN Communication IP discussed in D2.9, it is now possible within the RAIDER application to synthetize an hardware bitstream capable of work with a global clock of 200MHz. This clock increasing has been applied in the test for either testbed described in the previous section.

For the APE-Lab testbed setup, the integrals of both CPUs and FPGAs power profiles -depicted in the previous graphs – have been used to compute the energy-to-solution and energy efficiency values which are reported in Table 21, Table 22 and Table 23.

KPI	RAIDER @200 MHZ [2 FPGA, 1CNN]	RAIDER @200 MHZ [2 FPGA, 2CNNs]
time to solution [s]	4.644	2.332
throughput [events/s]	**581385.208**	**1162762.305**
energy to solution [J]	895.422 (370.672 FPGA)	449.622 (186.117 FPGA)
energy efficiency [events/J]	**3015.271 (7284.055 FPGA)**	**6005.044 (14506.937 FPGA)**

Table 21: [RAIDER] KPIs evaluated on the execution of the RAIDER application on a two Xilinx Alveo U200 FPGA system, processing 2.7M events with a global clock of 200 MHz

KPI	RAIDER @200 MHZ [3 FPGA, 2CNN]	RAIDER @200 MHZ [3 FPGA, 4CNNs]
time to solution [s]	2.332	1.161
throughput [events/s]	**1162762.305**	**1744120.135**
energy to solution [J]	703.460 (312.914 FPGA)	351.029 (156.593 FPGA)
energy efficiency [events/J]	**3838.171 (8628.577 FPGA)**	**7691.672 (17242.109 FPGA)**

Table 22: [RAIDER] KPIs evaluated on the execution of the RAIDER application on a three Xilinx Alveo U200 FPGA system, processing 2.7M events with a global clock of 200 MHz

KPI	RAIDER @200 MHZ [4 FPGA, 3CNN]	RAIDER @200 MHZ [4 FPGA, 6CNNs]
time to solution [s]	1.548	1.003
throughput	**2325478.642**	**2692072.654**

[events/s]		
energy to solution [J]	628.258 (283.615 FPGA)	407.512 (184.206 FPGA)
energy efficiency [events/J]	**4297.597 (9519.944 FPGA)**	**6625.572 (14657.541 FPGA)**

Table 23: [RAIDER] KPIs evaluated on the execution of the RAIDER application on a fourXilinx Alveo U200 FPGA system, processing 2.7M events with a global clock of 200 MHz

For the IDV-E testbed setup, the integrals of both CPUs and FPGAs power profiles, depicted in Figure 58, Figure 59 and Figure 60Figure 58: [RAIDER] FPGA power profiles (2 Alveo Xilinx U280 setup), have been used to compute the energy-to-solution and energy efficiency values which are reported in Table 24, Table 25 and Table 26. To note: the energy values reported are obtained - in the case of 3 and 4 FPGAs used - working on a combination of two Ampere Altra nodes with just one provided with the two-phase cooling system developed within the TEXTAROSSA project.

KPI	RAIDER @200 MHZ [2 FPGA, 3CNNs]
time to solution [s]	1.48956
throughput [events/s]	**1812622.012**
energy to solution [J]	216.170 (133.004 FPGA)
energy efficiency [events/J]	**12490.147 (20300.090 FPGA)**

Table 24: [RAIDER] KPIs evaluated on the execution of the RAIDER application on a two Xilinx Alveo U280 FPGA IDV-E node, processing 2.7M events with a global clock of 200 MHz

KPI	RAIDER @200 MHZ [3 FPGA, 6CNNs]
time to solution [s]	0.792
throughput [events/s]	**3409090.909**
energy to solution [J]	197.301 (105.489 FPGA)
energy efficiency [events/J]	**13684.675 (25595.171 FPGA)**

Table 25: [RAIDER] KPIs evaluated on the execution of the RAIDER application on a three Xilinx Alveo U280 FPGA configuration distributed over 2 IDV-E nodes, processing 2.7M events with a global clock of 200 MHz

KPI	RAIDER @200 MHZ [4 FPGA, 9CNNs]

time to solution [s]	0.554
throughput [events/s]	**4873646.209**
energy to solution [J]	165.277 (101.055 FPGA)
energy efficiency [events/J]	**16336.183 (26718.126 FPGA)**

Table 26: [RAIDER] KPIs evaluated on the execution of the RAIDER application on a four Xilinx Alveo U280 FPGA configuration distributed over 2 IDV-E nodes, processing 2.7M events with a global clock of 200 MHz

Finally, in Table 27, we consider the improvement factor over the KPIs baseline with RAIDER implemented on a single FPGA. For the comparison with both testbeds, we choose to report the values relatives to the higher throughput configuration obtained (since the throughput is one of the main goals of RAIDER application, developed to works in an HEP experiment with a trigger rate of 10MHz).

KPI	**Improvement factor over single U200 FPGA (2 CNN) setup**	
	APE Lab Testbed (4xU200, 6 CNN)	**IDV-E testbed (4xU280, 9 CNN)**
throughput	**4.884**	**8.841**
energy efficiency	**0.657**	**1.621**

Table 27: [RAIDER] Improvement factors of the KPIs for the tested designs over the baseline.

2.6.3 RESULTS ON TASK+STREAM INTEGRATION ON IDV-E

As part of Task 4.6 SW integration & Optimization we experimented how to integrate a RAIDER 6x6 kernel into OmpSs@FPGA framework. This proof-of-concept implementation was challenging as it implied mixing the task-based model that underpins OmpSs@FPGA with the stream model that is a natural approach to programming the RAIDER kernel. The objective of this exploration was to decide on the feasibility of this mixed model and the potential benefits that it could provide.
Several different implementations were tested. The main results are summarized in Figure 61 that shows the throughput in images processed per second (1 million image per second is equivalent to 1MHz) obtained by the different versions (please note the logarithmic scale of the vertical axis).

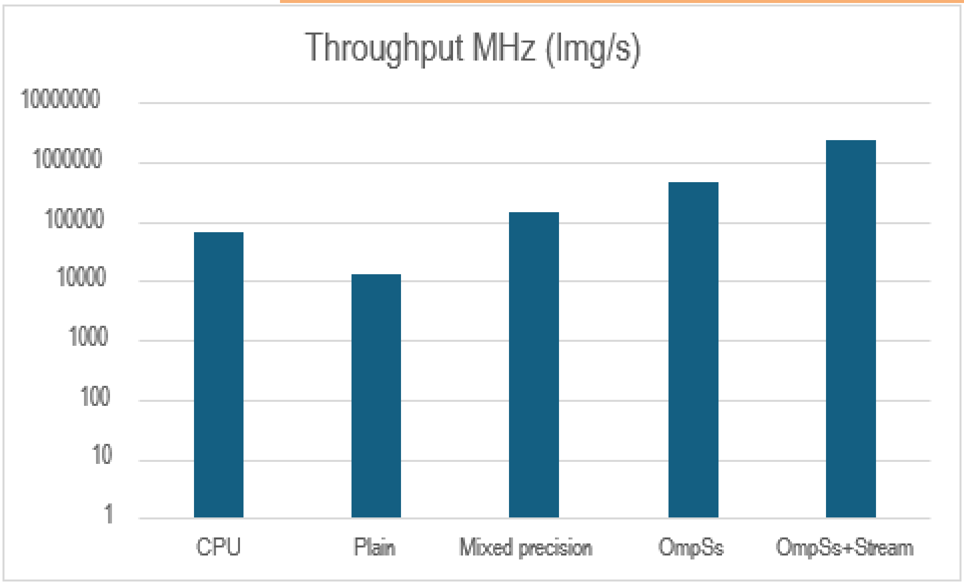


Figure 61: [RAIDER] Throughput with different OmpSs@FPGA implementations

As shown in Figure 61 there were several versions of RAIDER implemented in the OmpSs@FPGA framework. Figure 62 shows how the different versions change the work organization. Also, a description of the features of the main versions follows:

- CPU: This measures the performance of the RAIDER kernel optimized and executed in an Intel Xeon Silver 4208 CPU @ 2.1GHz. It has a throughput of 0.067MHz.
- Plain: This initial version implementation is a direct translation of the code executed in the CPU to the FPGA. In this initial version the OmpSs@FPGA framework is only used to facilitate the communication between the CPU driver program and the FPGA and to measure the FPGA performance. This plain version had a throughput of 0.013MHz so roughly a 5x slowdown over the CPU mainly due to the sequentialization of the image processing and the CPU slow sending of images to process to the FPGA.
- Mixed precision: In order to improve performance, an initial optimization taking advantage of the initial image pixel size was performed to use only the minimum number of resources in the FPGA. This optimization could not be applied to the CPU version as it can only exploit fixed size numbers (i.e byte multiples) but it was as simple as changing some type definitions in the FPGA code. The resulting code not only saved resources in the FPGA but also improved performance due to the lower latency of shorter bit operations and the minimized CPU-FPGA data transfer times reaching a throughput of 0.149 MHz. This version already has a 2x speedup over the CPU version.
- OmpSs: The third reported implementation used the FTS IP developed in Task 2.5 and OmpSs@FPGA to improve the kernel task latency increasing throughput significatively to 0.470MHz. This version managed tasks inside the FPGA but already in a sequential fashion.
- OmpSs+Stream: The final proof-of-concept version used the FTS to send tasks that pipeline the different images among the different processes in dataflow. This significantly improves performance as different images are in process in the same hardware kernel in the different processing steps. This implementation was able to reach 2.44MHz. This version is close to reaching the objective of 10MHz considering that under the OmpSs Framework it is trivial to divide the work into 4 or more kernels (as showed in section 3.12).

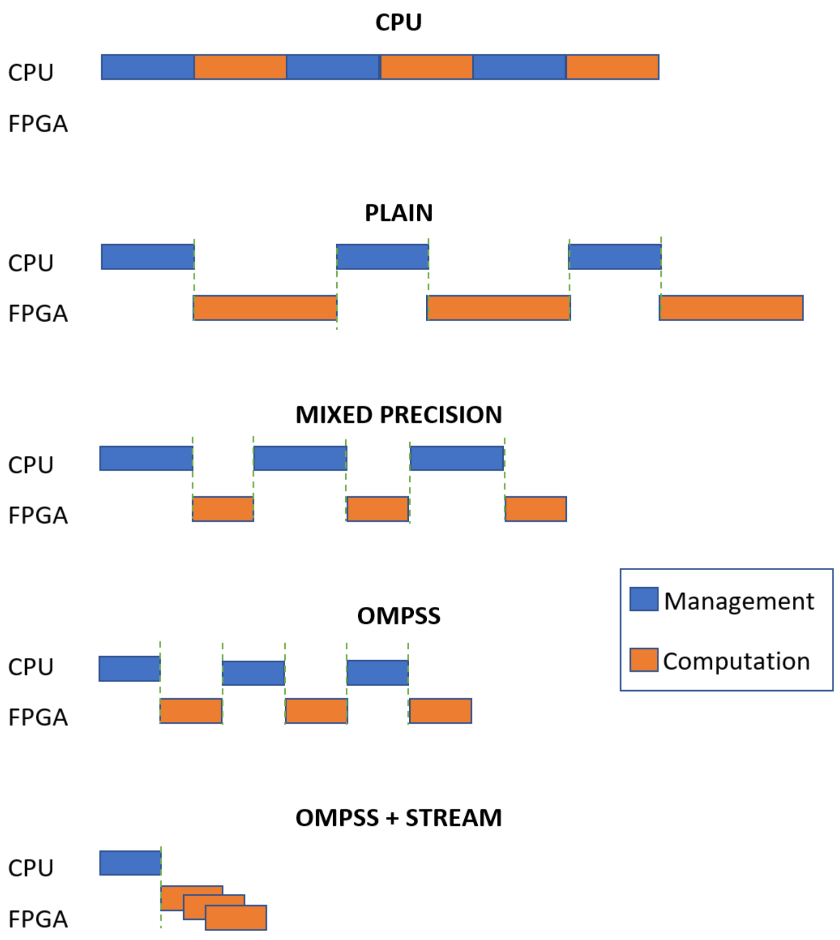


Figure 62: [RAIDER] Different OmpSs@FPGA implementations organization

Taking into consideration that the objective of the work presented in this section was limited to explore the feasibility of joining the task and stream models we consider that the results presented are highly successful as they demonstrate not only the aforementioned feasibility but show a promising venue to leverage it to achieve some significant performance goals. Giving this results more work should be devoted in this direction to integrate the changes into the automatic tools to facilitate development and to evaluate ways to further improve performance and even better mix both models in more intuitive ways.

2.7 TNM – INFN

The tensor network methods (TNM) application combines multiple simulation frontends for simulation quantum systems. For TEXTAROSSA, we consider the following sub-applications:

- Quantum matcha TEA: gate-based quantum circuit emulator for digitized quantum circuits (analyzed in deliverable D6.2).

- Quantum green TEA: solver for the Schrödinger equation or Lindblad equation; within this analysis, we restrict ourselves to finding the ground state of a system.

2.7.1 Towards mixed precision

In the current development goals, mixed precision algorithms are not on the agenda for the tensor network methods. Nonetheless, we take TEXTAROSSA as motivation to compare different precisions for the calculations of the TNM. This data helps us to decide in the future if mixed precision can be beneficial for our application. In deliverable D6.2, we already presented data for Quantum matcha TEA and single precision versus double precision complex numbers for a quantum circuit. Now, we turn to quantum green TEA, where we added the possibility to run simulations with real data types and switch to higher precision towards the end of search; real data types are suitable as long as the Hamiltonian is also real. The ground state energy of a system with 128 qubits can be calculated via the Jordan-Wigner transformation (E=-162.6123001775688(7), J=1.0, g=1.0).

For the comparison, we use the following configuration:

- TEXTAROSSA-node Dibona (Atos)
- Quantum green TEA (Version pre_v0.3.17)
- single-cpu simulation
- likwid 5.2.2 and per-cpu energy measurement

Figure 63 and Figure 64 show the same trend for runtime and energy consumption where the double complex data point (black) serves as reference for simulations before the implementations done within TEXTAROSSA. The colormap shows how many of the last sweep are in double precision; not all sweeps have to executed based on an exit criterion. We observe that using only single-precision sweeps leads to an increased error. Already two double-precision sweeps are sufficient to bring the error on par with the best simulations. The optimal setting here is five single-precision sweeps followed by potentially five double precision sweeps, while actually exiting after the ninth sweep.

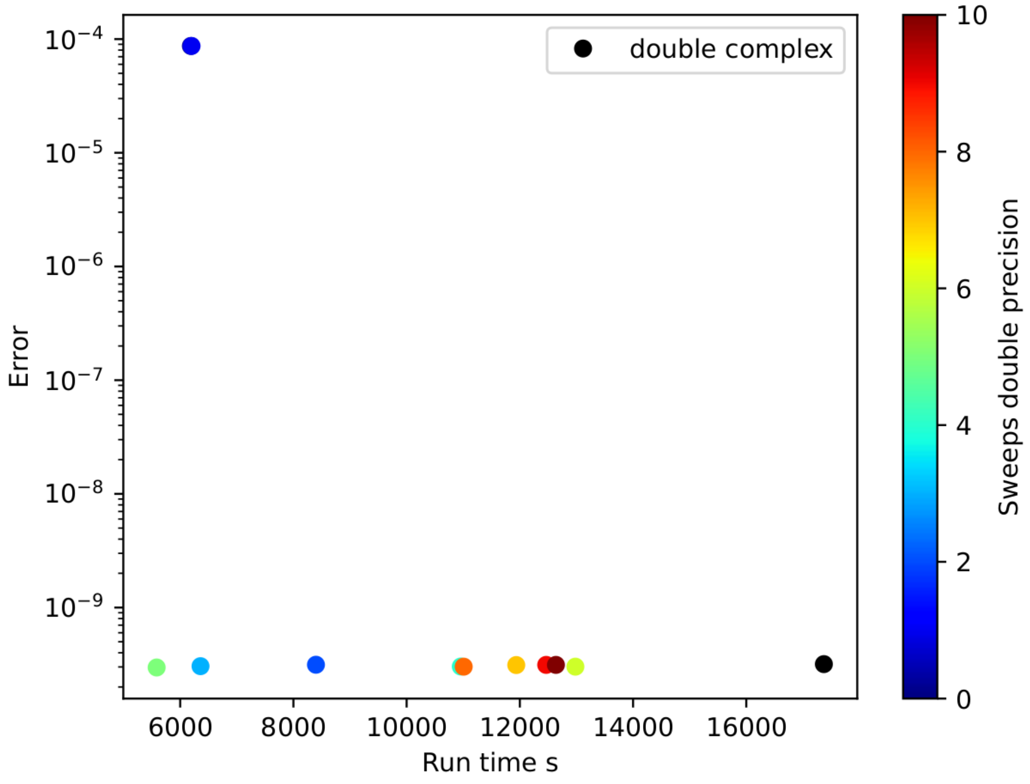


Figure 63: [TNM] Execution time vs error varying the number of final sweeps in double precision

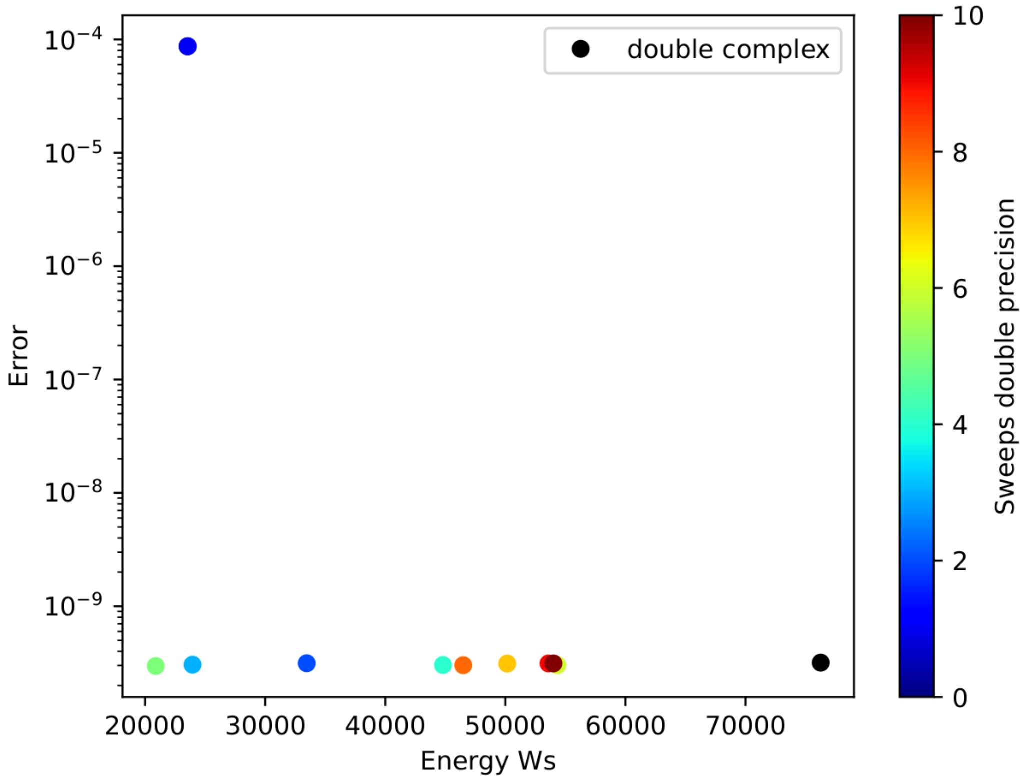


Figure 64: [TNM] Energy consumption vs error varying the number of final sweeps in double precision

In summary, the ability to run with different precisions and increase during runtime the precision is a useful path for quantum green TEA and future implementation, e.g., an automatic increase of the precision independent of the user input.

These changes have been implemented on the Fortran level of the quantum green TEA library.

2.7.2 Hybrid simulation: data parallelism via MPI versus parallelising a single simulation

One a single node as targeted by TEXTAROSSA, one can find the optimal path between running a batch of simulation in a hybrid MPI-openmp/threading scenario. For the quantum green tea application, we focus on threading via the MKL library. We use the following setup:

- Justus2 cluster (bwHPC): 2xIntel Xeon 6252 Gold with 2x24 cores; we run quantum green tea v0.3.29 on Justus2.
- Batch of 96 identical simulations, overloading node by one MPI thread as one MPI thread is not executing any computational workload in the master-worker approach.
- We have N workers, an total time T for all 96 simulation (wall-time), the time t for a single simulation, and the speedup S for a single simulation. The number of MPI threads is 48 / N.
- The data points are for the quantum Ising model in one dimension at the quantum critical point.

TNM hybrid simulation MKL threads	Time total T [mm:ss]	Time t for single simulation [mm:ss]	Speedup S for single simulation
1	6:16	3:08	Reference
2	6:17	1:34	1.99
3	7:26	1:14	2.5
4	7:35	0:57	3.3
6	**7:43**	**0:39**	**4.9**
8	18:55	1:11	2.7
12	21:38	0:54	3.5
16	33:30	1:03	3.0
24	51:01	1:03	2.9
48	134:44	1:24	2.2

Table 28: [TNM] Hybrid simulation with MPI and threading via MKL library. We use simulation of the quantum green tea library out of the Quantum TEA suite.

We observe two main trends in Table 28. To minimize the total simulation time, MPI is always the best approach at constant resources. The speedup of the MKL library peaks for this setup at 6 threads. With scalable resources, running 96 MPI threads with each 6 MKL threads across multiple nodes minimizes the wall-time. The minimum of time t as a function of MKL threads might be problem dependent and platform dependent, e.g., which is the size of the generalized matrix problems to be solved in the simulation or the NUMA modes implemented on the architecture.

2.7.3 Device comparison CPU versus GPU

Motivated by the mixed precision results on the Fortran level, we implemented the ground state search of quantum green TEA on the Python side of our library supporting switching precision during the sweeps as well as GPU support. We consider a two-dimensional quantum Ising model and its ground state search via a Tree Tensor Network (TTN). We run on CINECA's Leonardo, single-core for CPU simulations (Intel Xeon 8358 CPU, with 32 cores running at 2.6 GHz), single GPU (NVIDIA custom Ampere GPU, 64GB HBM2) and highlight in the following three figures three aspects:

1. For larger system sizes, the GPU gives a benefit in terms of speedup, see Figure 65;
2. both devices show a speedup when moving sweeps, i.e., iterations of our search, to single precision, see Figure 66;
3. except all sweeps being done in single precision, the same precision for the ground state is reached (lower ground state energy is better), here shown for at least two double precision sweeps at the end, both for CPU and GPU, see Figure 67.

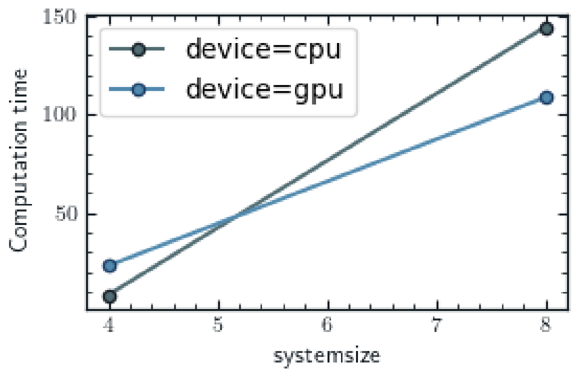


Figure 65: [TNM] Execution time vs system size

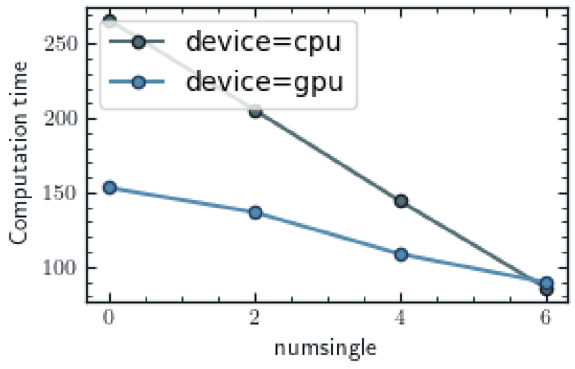


Figure 66: [TNM] Execution time vs number of sweeps in single precision

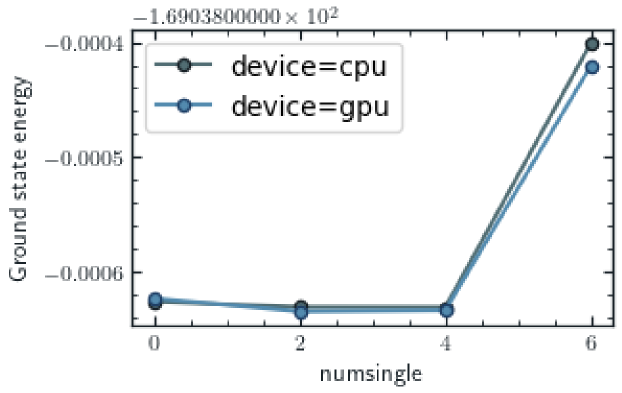


Figure 67: [TNM] Ground energy state (lower is better) vs number of single precision sweeps

Algorithmic improvements for sampling

In addition to the above studies on aspects of different precision and parallelization, we have undertaken some algorithmic improvements which lead to an improvement of the energy consumption of the library via the improved computation time. The following example is on the problem of sampling the classical outcomes of a quantum state represented as a tensor network, which are discussed in detail in M. Ballarin's work [11]. Out of the various example, we pick the analysis of a tree tensor network representing the ground state of the quantum Ising model while we try to reach a given threshold of probability coverage. Our new algorithm has shown a speedup, but we verify that the speedup is also reflected in the energy consumption on Dibona node. The key data are:

- TEXTAROSSA Dibona node (Atos)
- qtealeaves python library
- single-core simulation
- likwid 5.2.2 and per-core energy measurement

Figure 68 and Figure 69 the speedup of the new algorithm in computation time and the same ratio in terms of energy consumption.

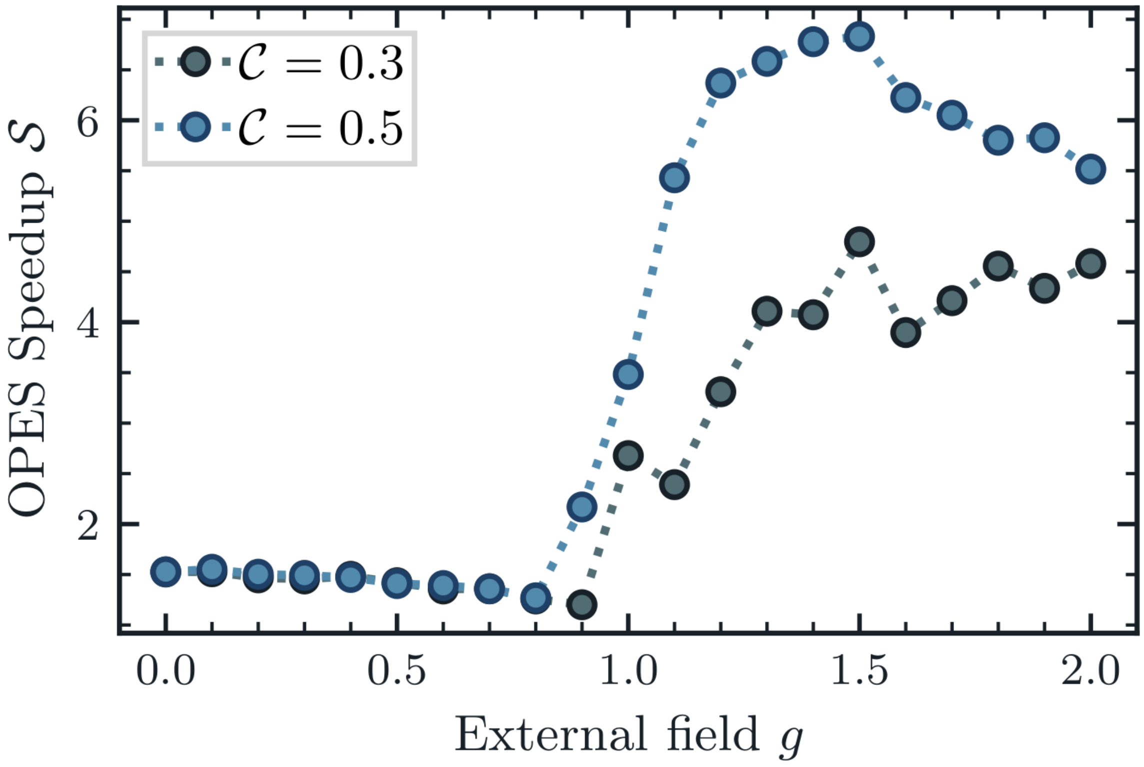


Figure 68: [TNM] Speedup after algorithmic improvements

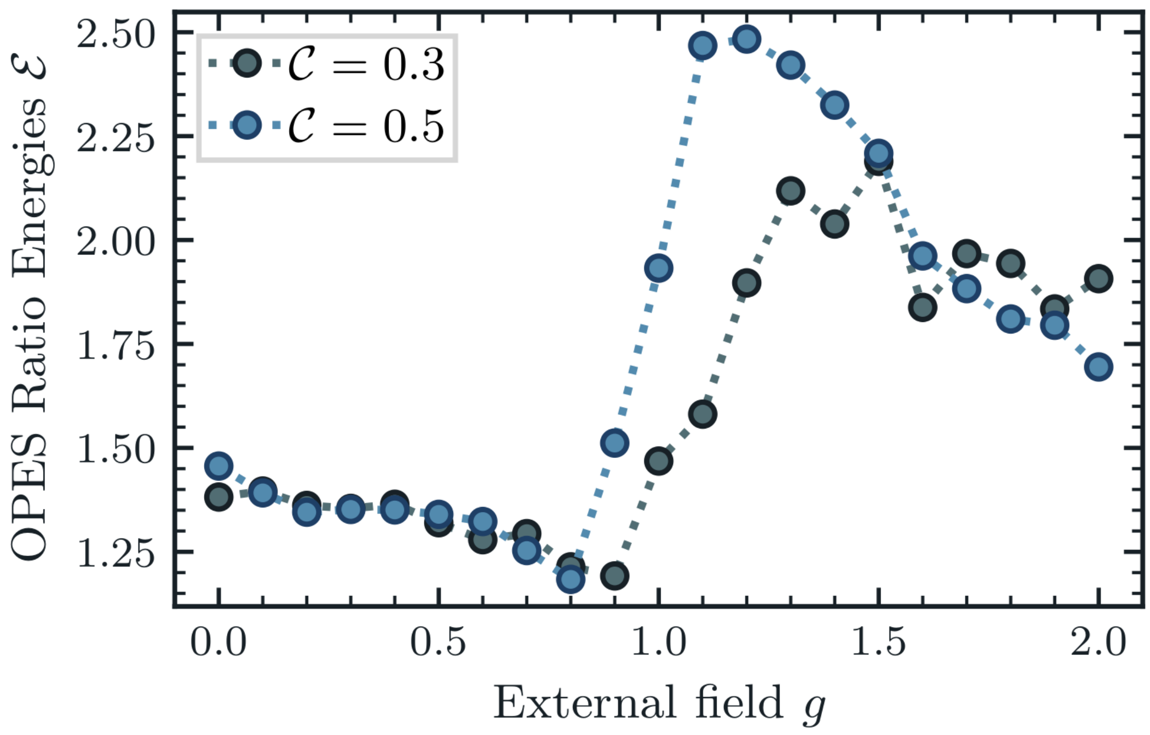


Figure 69: [TNM] Improvement ratio in energy efficiency after algorithmic improvements

2.7.4 Algorithmic improvements for Hamiltonian representation

We were furthermore able to optimize the algorithm how we represent a Hamiltonian (matrix) and apply it to a state (vector), i.e., a generalized matrix vector multiplication. This application is the foundation of the ground state search and time evolution within quantum green TEA. The results show the relative computation time of a ground state search with respect to the best method, i.e., the icTPO (indexed compressed tensor product operator), see Figure 70. The tensor product operator (TPO) and our starting point before the TEXTAROSSA project are about three times slower in the example system; the indexed tensor product operator (iTPO) and an alternative algorithm running as reference point (sparse-matrix product operators (SPO) and indexed SPO (iSPO)) are about two times slower than the icTPO. The results are obtained for random all-to-all two-body interactions in a quantum Ising type of Hamiltonian with additional randomized local fields. The simulations run on the Leonardo machine at CINECA, as single-CPU simulation. Future steps are to benchmark how algorithmic improvement develops in a parallel algorithm.

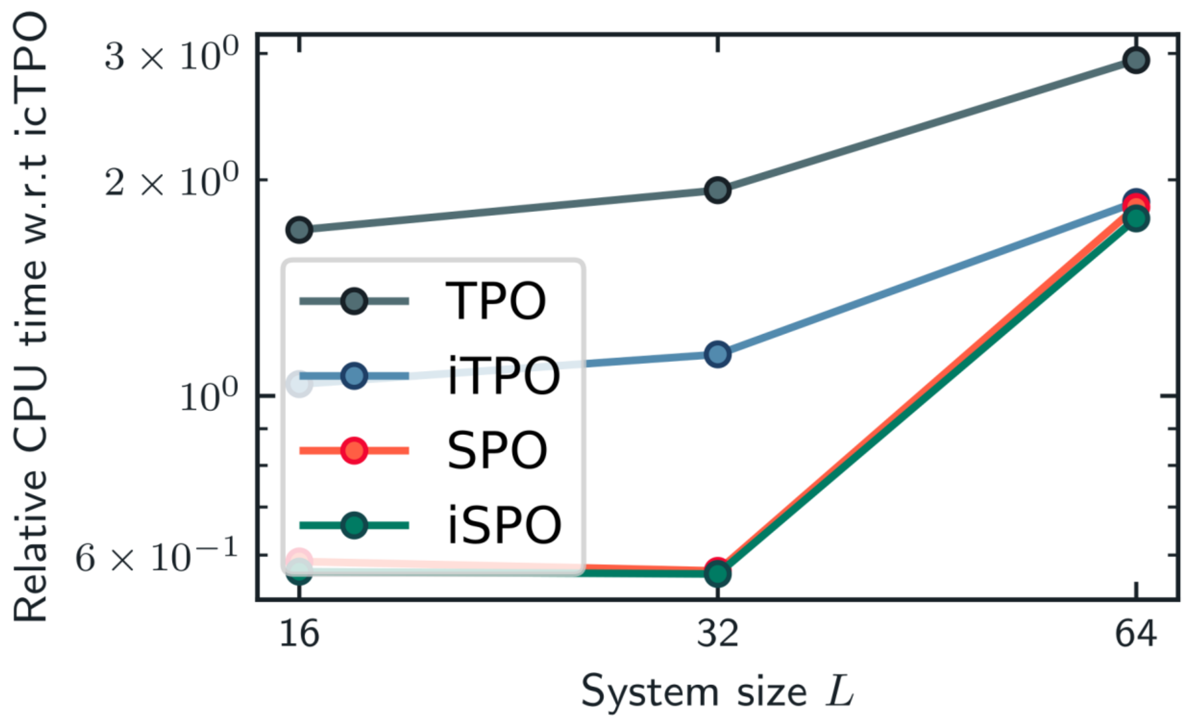


Figure 70: [TNM] Relative CPU time w.r.t icTPO

2.8 ScalFMM (Mathlibs-INRIA)

The Fast Multipole Method (FMM) is a highly efficient algorithm for computing long-range forces in N-body simulations, which are prevalent in fields such as astrophysics, molecular dynamics, and electrostatics. The traditional FMM algorithm reduces the computational complexity of calculating pairwise interactions from O(N2) to O(NlogN) or even O(N), depending on the implementation and the desired accuracy.

TBFMM - a fork of ScalFMM - is high-performance task-based FMM implementation where the algorithm is decomposed into a series of discrete tasks, each representing a portion of the computation. These tasks are defined with clear dependencies, forming a Directed Acyclic Graph (DAG) that represents the execution flow of the entire computation. This DAG includes tasks for operations such as particle-to-particle interactions, particle-to-multipole translations, multipole-to-multipole translations, and more, depending on the specific version of the FMM algorithm being used.

In the TEXTAROSSA project, we developed a new scheduler to decide how the tasks should be distributed on the different processing units. Those processing units are usually heterogeneous with a mix of CPUs/GPUs (IDV-A) or CPUs/FPGAs (IDV-E). Our scheduler is detailed in WP4, and WP6 is tied to its use and analysis.

We provide the results in Figure 71. We executed a simulation of 1 million particles on two hardware configurations, Intel CPU + NVIDIA V100 and AMD CPU + NVIDIA A100 (similar to IDV-A). We compared our scheduler against two state-of-the-art alternatives. Our scheduler, MulTiPrio, demonstrates very good results. In a more in-depth study, we showed that our scheduler is especially competitive when the tasks are irregular (i.e., of various granularities).

Thus, MulTiPrio represents an innovative approach to dynamic task scheduling that significantly benefits the execution of TBFMM by reducing the overall computation time while maximizing hardware utilization.

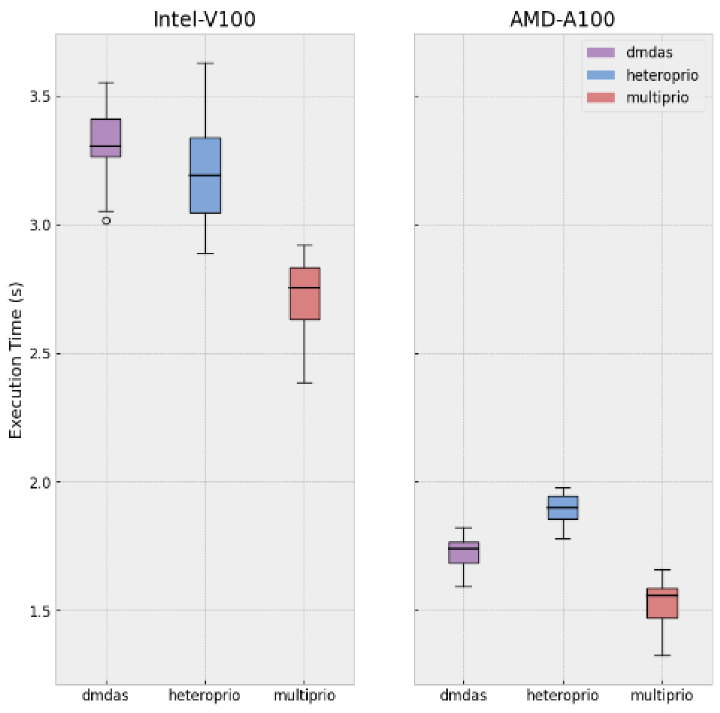


Figure 71: [FMM] Performance results for TB-FMM on two hardware configurations

The execution time is provided (the lower the better)

2.9 Chameleon (Mathlibs-INRIA)

Chameleon is a software library designed to optimize the execution of dense linear algebra computations on heterogeneous computing systems. It provides high-performance implementations of various linear algebra operations, adapting to the underlying hardware to achieve efficient execution. The library leverages task-based programming models to decompose complex operations into smaller tasks, facilitating their scheduling and execution across different types of processing units, such as CPUs and GPUs. This approach allows Chameleon to dynamically balance the workload and utilize available hardware resources effectively.

Within the Chameleon library, key kernels such as GEQRF, POTRF, and GETRF are implemented to support a range of linear algebra computations:

- GEQRF (QR Factorization): This kernel computes the QR factorization of a matrix. QR factorization is a process that decomposes a matrix AA into the product of an orthogonal matrix QQ and an upper triangular matrix RR. The GEQRF kernel is crucial for solving systems of linear equations, eigenvalue problems, and many other applications in scientific computing. In the context of Chameleon, the GEQRF operation is broken down into tasks that can be executed in parallel, exploiting data locality and minimizing data movement for improved performance on heterogeneous systems.
- POTRF (Cholesky Factorization): The POTRF kernel performs the Cholesky factorization of a symmetric positive-definite matrix AA, decomposing it into the product of a lower triangular matrix LL and its transpose LTLT. Cholesky factorization is particularly useful for solving linear systems where the matrix is symmetric and positive-definite, such as in optimization problems and numerical simulations. Chameleon optimizes the execution of POTRF by scheduling tasks efficiently across available resources, taking advantage of the specific characteristics of the matrix to enhance performance.
- GETRF (LU Factorization): This kernel computes the LU factorization of a general matrix. LU factorization decomposes a matrix AA into the product of a lower triangular matrix LL and an upper triangular matrix UU, possibly with a permutation matrix PP due to partial pivoting. GETRF is a fundamental operation in linear algebra, enabling the solution of systems of linear equations, inversion of matrices, and computation of determinants. In Chameleon, the GETRF operation is managed through a task-based approach, allowing for concurrent execution of tasks and efficient use of heterogeneous computing resources.

As for ScalFMM, we used our scheduler on Chameleon and provided the result in Figure 72. We compared MulTiPrio against DMDAS and heteroprio, on Intel CPU + NVIDIA V100 and AMD CPU + NVIDIA A100 (similar to IDV-A). Our new scheduler is competitive with DMDAS but is slightly slower. The main reason is that DMDAS use high-tuned priorities provided by the developers (which for example inform the scheduler about the critical path without the need of analysis). Our scheduler is able to use them too, but in the given results we use a version where our scheduler tries to find the critical task autonomously.
We provide two execution traces, see Figure 73 and Figure 74, to illustrate the difference. We observe that when priorities are utilized, the number of ready tasks is well-controlled, meaning that the most critical tasks are computed first.

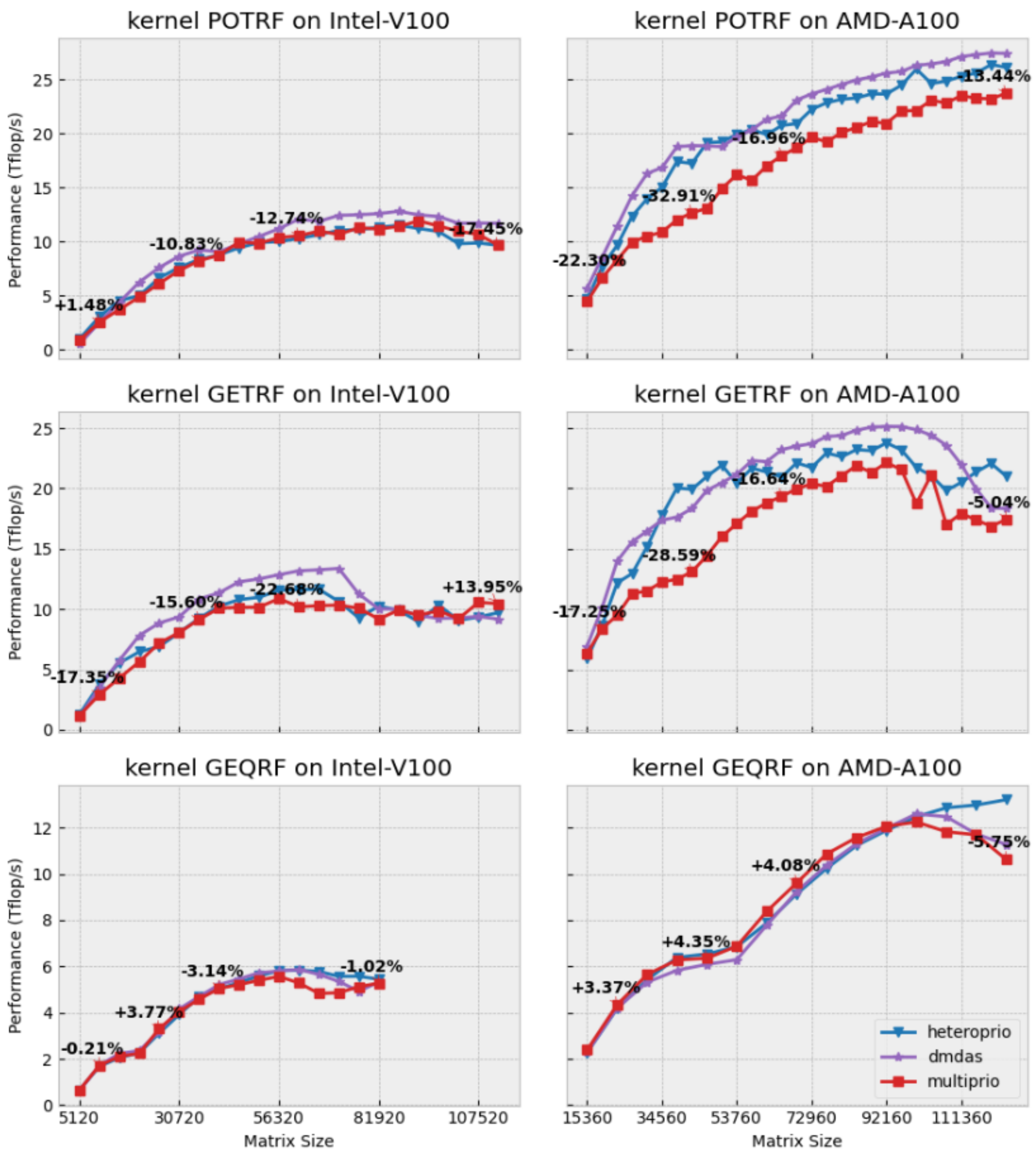


Figure 72: [CHAMELEON] Performance results for Chameleon on two hardware configurations
Flops per second are provided (the higher the better

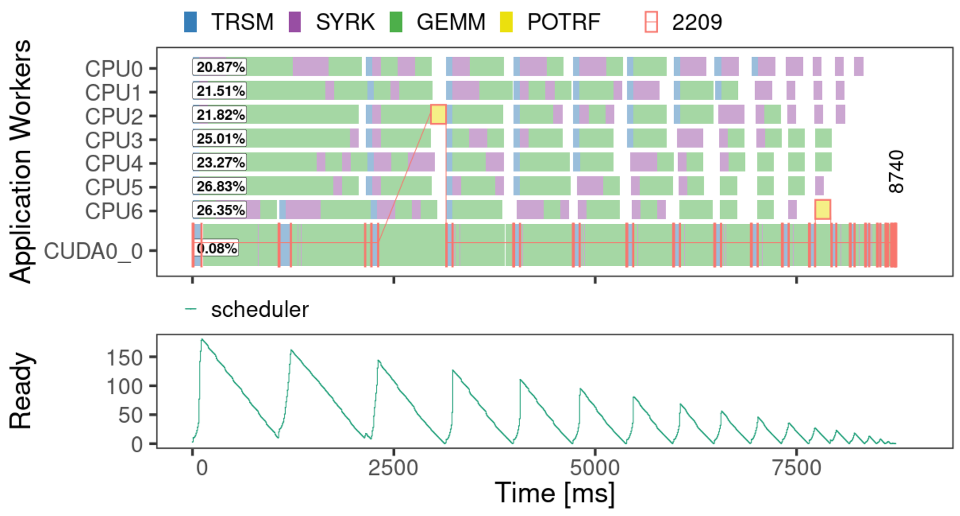


Figure 73: [CHAMELEON] Execution trace (emulation) for a Cholesky factorization without using user's priorities

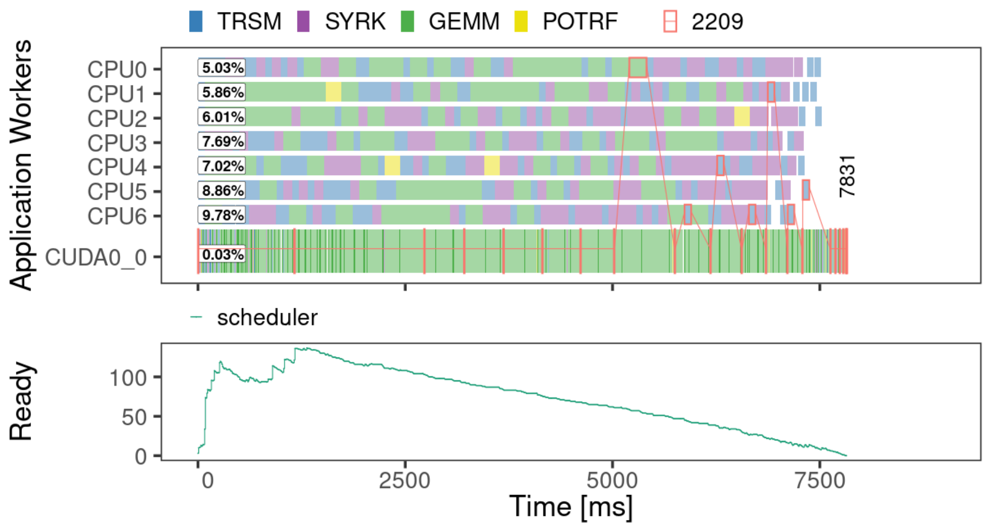


Figure 74: [CHAMELEON] Execution trace (emulation) for a Cholesky with user's priorities

2.10 UrbanAir - PSNC

As described in previous deliverables, selected UrbanAir kernels were adapted to benefit from usage of heterogeneous resources, exploiting CPUs and NVIDIA GPUs. While results are provided for strong scalability, the usual scenarios for assessing air quality runs on domain of size exceeding the memory resources of a single GPU, therefore we focus on weak scalability. The toolchain includes C++ compiler, the CUDA toolkit, threads (OpenMP) for shared-memory parallelization (single node) and MPI for data exchange between GPUs, all available at IDV-A platform. To monitor energy consumptions of kernels running on GPUS, GPower project tool was used, also available on IDV-A TEXTAROSSA platform. As previously, the tests were conducted on PSNC Altair machine equipped with NVIDIA V100 and the newly available IDV-A equipped with NVIDIA H100.

The UrbanAir-gcrk starts with solver initialization, followed by a guess of initial wind velocity, estimation of pressure and initialization of boundary condition. Eventually, iterative solver is started where for each iteration reduction and preconditioner (to speedup solver) subroutines are called, and Laplacian operator is solved. Every sub-routine is provided with a separate implementation for CPU and GPUs, so that within a compilation user can decide whether to run on CPUs or GPUs or use a mix of them.

The global domain is divided between the workers, and each receives a 3D block called subdomain, which size is determined by the hardware used. The domain decomposition is done manually before the compilation for additional compiler optimizations. On a single node with CPUs only, OpenMP is used to exploit shared memory. To exchange data between the nodes, and between GPUs, MPI paradigm is used. On order to achieve the best performance on GPUs, the size of subdomain on GPUs is the maximum possible (memory constraint). The implementation allows to execute simultaneously on CPUs and GPUs – on GPUs maximum possible size of domain is assigned, while the remaining is solved on CPUs – the latter need to be carefully chosen so that parts executing on GPUs do not have to wait for iteration done on CPUs. However, on currently available fat nodes with multiple GPUs, the most efficient execution is to use accelerators and use CPU only to handle communication between GPUs. Once the domain is divided into subdomains, it is then additionally divided into blocks for optimal cache utilization. As of communication, it is overlapped with computation: computation of boundary conditions is separated from computation of inner domains, so that the data at boundaries is exchanged with neighbors while the inner domain is still computed.

UrbanAir-gcrk iterates over whole domain doing stencil computations. Every subdomain is assigned to exactly one GPU and after each iteration the data between subdomains is exchanged using additional HALO cells (boarders of the subdomain). To increase performance on NVIDIA V100 and newly emerged H100 GPUs cards, and to support wind-flow-specific settings, additional improvements were provided: i) additional boundary conditions /data were placed in shared memory to speedup, ii) further optimization of Laplacian operator implementation for GPUs was introduced, iii) number of communication in *rhsdiv* kernel was limited, iv) further optimal size of the block of threads within subdomain for each sub-kernel is selected based on some auto-tuning (kernel compile – run – compare results). After each iteration, a global sum of variables is calculated – reduction algorithm was improved to benefit from shared memory in multi-GPU environment. With these changes provided, further up to 10% decrease of time-to-solution is observed.

Adaptation to GPUs and optimization is a game changer, not trivial though, in terms of time-to-solution decrease, as reported already in D6.2. For applications were data to be computed can be equally divided between the workers, it is easier to adapt as the workload for each GPU is the same and developer needs to focus on providing implementation for an accelerator and how to just exchange data between subdomains. As the previous results and those in following subsections have described executions on GPUs are superior to CPUs, though the best efficiency can be achieved only with a GPU fully loaded with data to be computed.

2.10.1 Multi-node environment

To test UrbanAir-gcrk in multi-node environment, PSNC's Altair HPC system was used, with up to 6 nodes available, each equipped with 8 NVIDIA V100 GPU cards Two tests were conducted based on strong and weak scalability. Figure 75 presents the strong scalability for 5M grid points problem size. With more GPUs added, the number of iterations solved within one second increases. Provided optimizations allows to reach up to 11% increase in performance, which eventually drops when more GPUs are used, because the problem size per GPU becomes too small to efficiently utilize GPU. Figure 76 presents weak scalability for a fixed problem size – 5M grid points per GPU. Although the number of iterations solved initially drops when more GPUs are added, it then oscillates close to 40 for the remaining GPUs, making it possible to solve 240M grid points problem within the same amount of time.

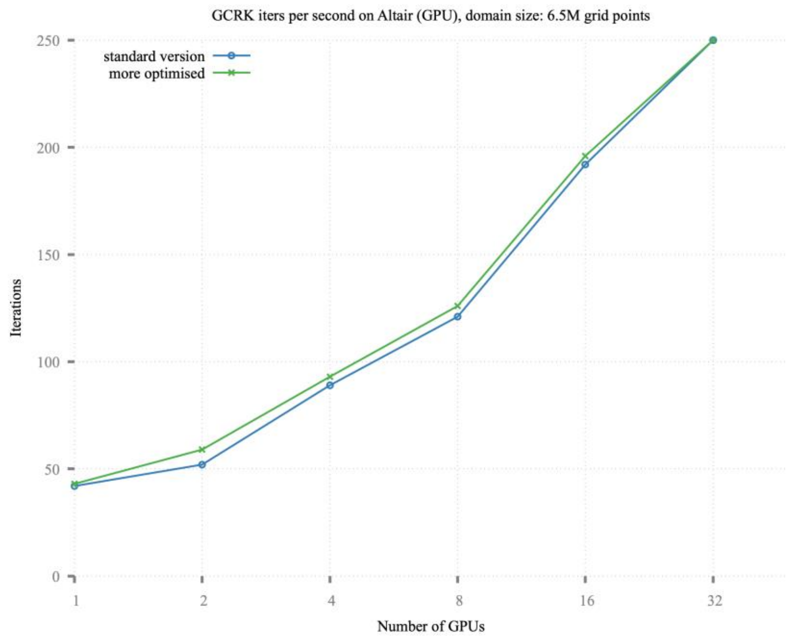


Figure 75: [UrbanAir] Iterations per second (problem size: 6.5M)

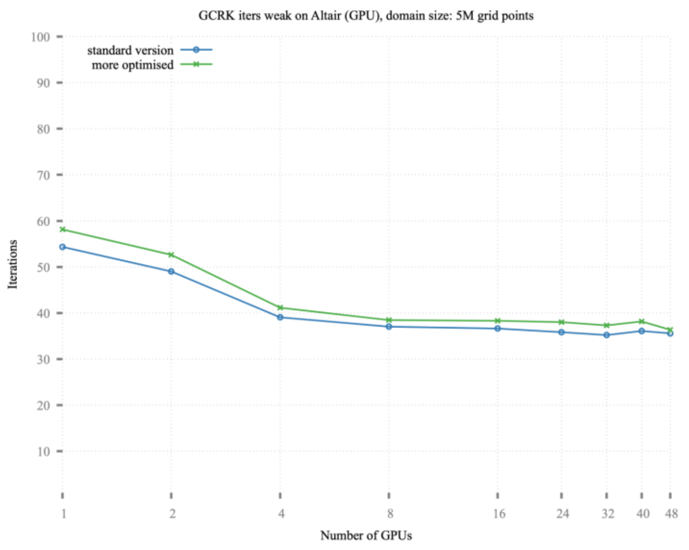


Figure 76: [UrbanAir] Iterations per second (5M per GPU)

The 5M grid points per GPUs is suboptimal because the GPU is not fully loaded with data it computes. Figure 77 presents comparison of achieved speedup between 5M and 50M grid points (maximum size on NVIDIA V100 due to memory constraints). It is clearly visible that the larger problem size is, the better speedup can be achieved, thus obtaining results in a shorter amount of time. Similarly, weak scalability of UrbanAir-gcrk is better with GPUs fully loaded with data. Figure 78 presents efficiency comparison between 5M and 50M grid points per GPU. While efficiency is better for a maximum executable problem size, it decreases with more GPUs used, and thus the time-to-solution becomes slightly longer. Nevertheless, with GPUs we are able to run UrbanAir-gcrk over much larger areas within a reasonable amount of time.

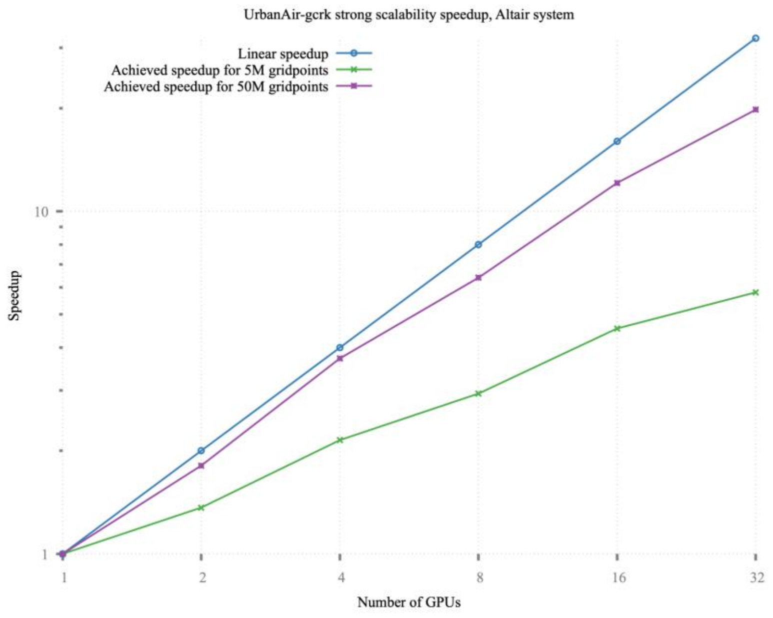


Figure 77: [UrbanAir] Speedup comparison for different domain sizes

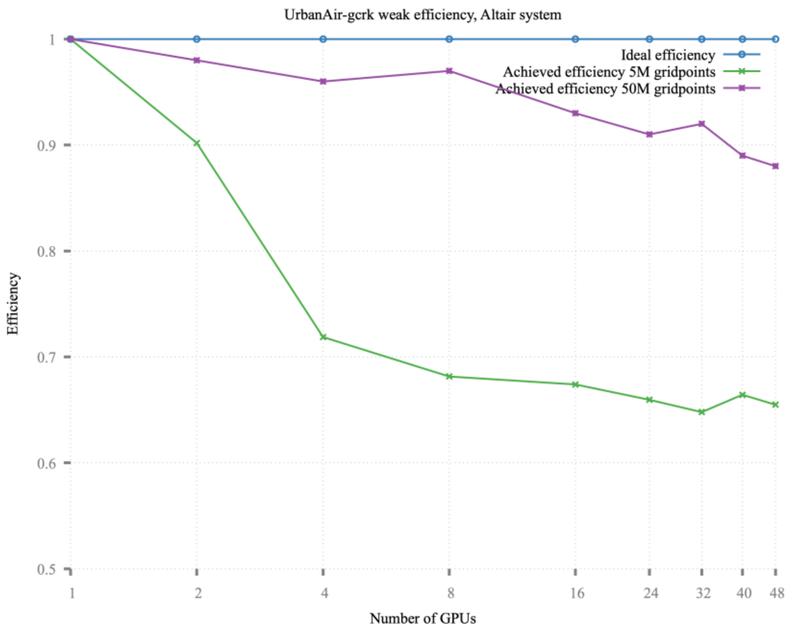


Figure 78: [UrbanAir] Efficiency (weak speedup) comparison for different domain sizes

2.10.2 IDV-A results

Table 29Table 29: [UrbanAir] KPIs overview presents KPIs for UrbanAir-gcrk, which are then discussed in the following subsections. In previous deliverable, D6.2, initial baseline results were discussed, comparing execution on CPUs vs GPUs. PSNC Altair system and Dibona (predecessor of IDV-A) were used.

KPI for computational efficiency	KPI for energy
- Time-to-solution - (strong and weak) speedup - number of iterations / second	Iterations / Watt

Table 29: [UrbanAir] KPIs overview

2.10.2.1 Computational efficiency

NVIDIA H100 GPU cards available at IDV-A node allows to run larger UrbanAir-gcrk problems, comparing to V100 available on Altair PSNC system: 59M grid points instead of 50M. More important difference comes with performance, for the same problem size (50M) the H100 GPU cards offer 2x decrease in execution time comparing to V100 GPU, see Figure 79.

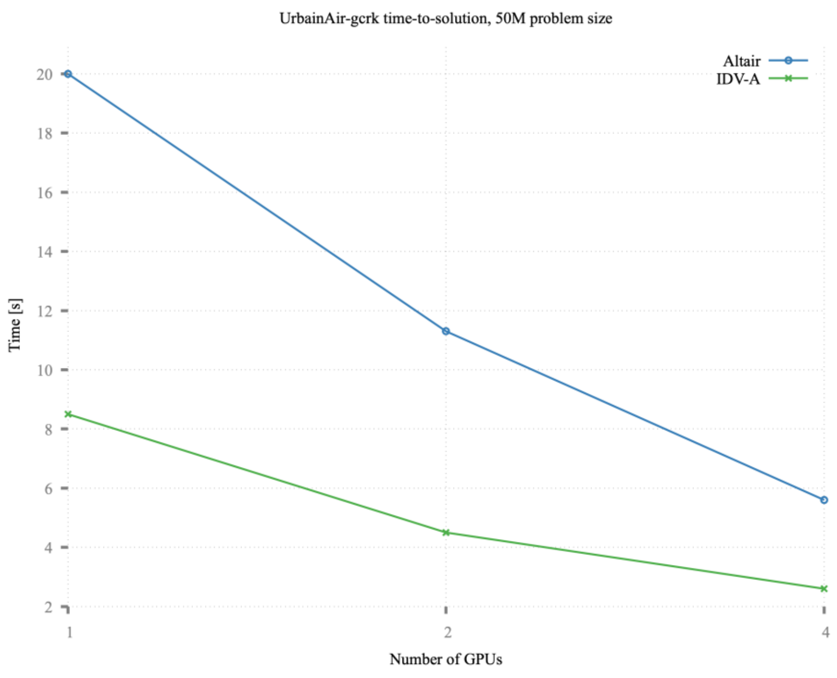


Figure 79: [UrbanAir] Time-to-solution on GPUs: V100 vs. H100

Figure 80 compares speedup of UrbanAir-gcrk across different systems, configurations and code optimizations. Dibona is the former IDV-A system, results for which were presented in D6.2. On final IDV-A, three different configurations are compared:

- IDV-A with traditional cooling system (wo-2phase)
- DV-A with two-phase cooling system installed (w-2phase),
- IDV-A with two-phase cooling system on which optimized version of UrbanAir-gcrk was run (w-phase-opt).

The speedup characteristic is much the same for each configuration, with slightly decreases with more GPUs being used. The worse result is observed for w-phase-opt, while the best speedup is achieved on Dibona and wo-2phase. While w-phase-opt provides the worse speedup, it achieves the best iterations/s KPI, see Figure 81. It is worth mentioning that thermal controller of the 2-phase cooling is actively monitoring GPUs for its load to ramp-up their frequency whenever required. Therefore, it provides some overhead, or rather very slight performance decrease of applications running on GPUs. In case of UrbanAir-gcrk, it impacts time-to-solution the more, the more GPUs are used. Provided optimizations mitigates this impact. Compared to the initial version of IDV-A, Dibona, 10% increase in number of iterations per second is observed.

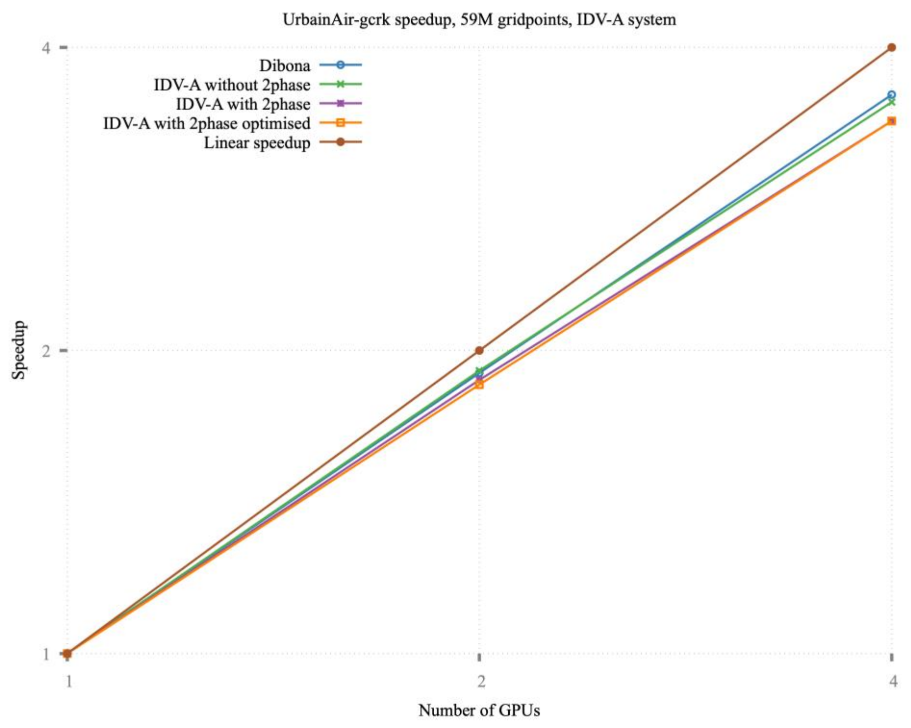


Figure 80: [UrbanAir] Strong speedup comparison for 59M grid points

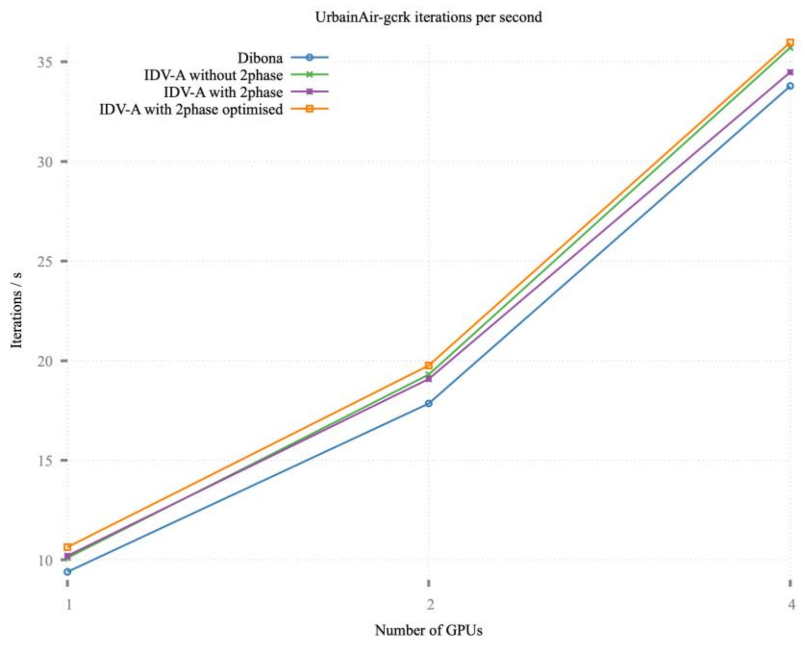


Figure 81: [UrbanAir] Iterations per second for 59M grid points

2.10.3 Energy efficiency

To measure energy efficiency, the methodology proposed in D6.2 is used. Figure 82 details comparison of energy consumption of UrbanAir-gcrk run for 59M grid points domain (strong scalability) for different number of GPUs (rows in Figure) and three different systems (columns in Figure): Dibona (previous version of IDV-A, left column), IDVA-wo-2phase (IDV-A without 2 phase cooling system, left middle column) and IDVA-w-2phase (with two phase cooling system installed, right middle column) and IDVA-w-phase-opt (two phase cooling system installed, additional optimizations of kernels). For each system, adding more GPUs results in less power consumption of each, which is due to fewer grid points to be computed by each GPU. Dibona, IDVA-wo-2phase and IDVA-w-2phase have the same characteristic of energy consumption – after reaching the highest level of energy utilization, power draw remains more or less constant during the entire execution. It is worth noticing that power draw during execution is 7% higher compared to Dibona. In contrast, additional optimizations introduced (IDVA-w-2phase-opt) provides different power characteristics. It starts with reaching the highest level of energy consumption and almost immediately the power draw drops to an even lower level than observed on Dibona. Taking this into account, and that time-to-solution is shortened when running on IDVA-w-2phase with optimized code, the consumed power should be the smallest among others.

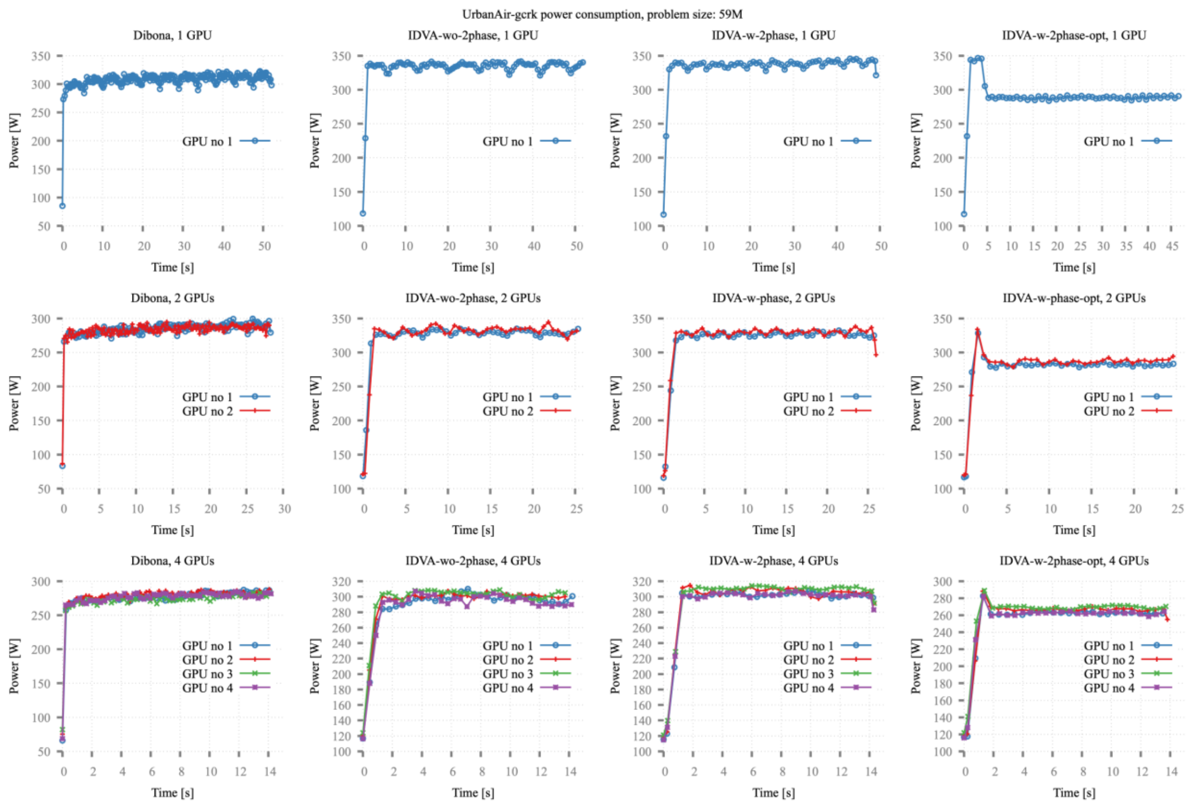


Figure 82: [UrbanAir] Energy usage characteristic for 59M grid points

Figure 83 presents comparison of energy used for running UrbanAir-gcrk on different systems. As discussed above, runs on Dibona were the most energy efficient until more optimizations were introduced to UrbanAir-gcrk. Please note that only GPU power usage being used in computation is taken into account. The CPU power usage is neglected as the CPU is only used to orchestrate data exchange between GPUs. Figure 84 presents energy usage when all GPUs available at the node are taken into consideration, i.e. for a single GPU run, energy usage of all 4 GPUs available are summed. As one can expect, the most energy efficient execution is when all available GPUs are used for computations. It is important to notice that introduced optimizations resulted in the most energy-efficient run on IDV-A.

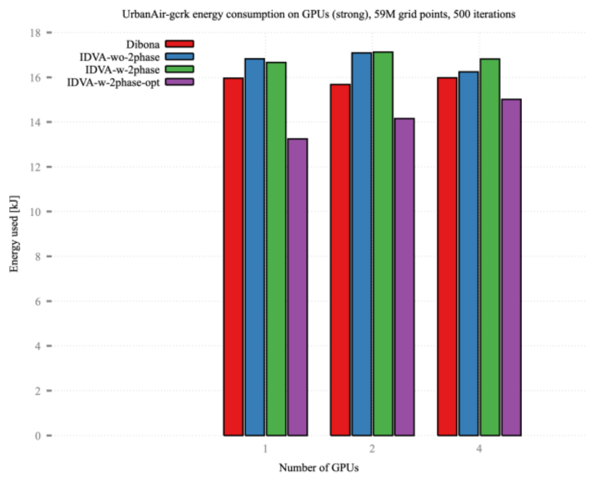


Figure 83: [UrbanAir] Energy consumption for 59M grid points, only used GPUs accounted.

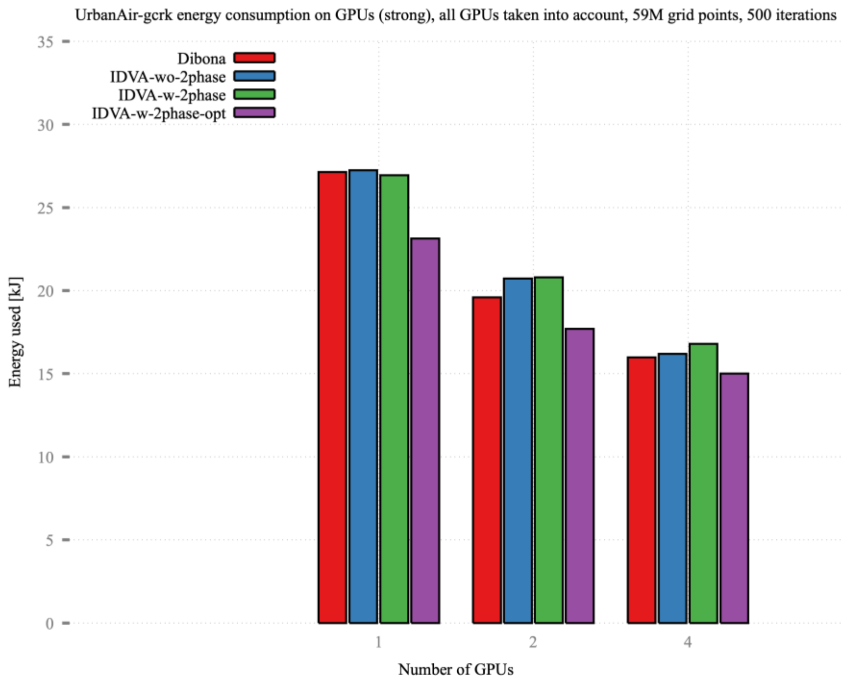


Figure 84: [UrbanAir] Energy consumption for 59M grid points, unused GPUs accounted.

UrbanAir-gcrk scales weakly very well when entire GPUs is occupied by data, as discussed in previous subsections. Figure 85 compares IDV-A with and without optimization with respect to time-to-solution (top), and energy consumption (bottom) for a fixed problem size per GPU (59M grid points). With the new version, 4% decrease in time-to-solution is observed, and 22% decrease in energy consumed (when all 4GPUs are used).

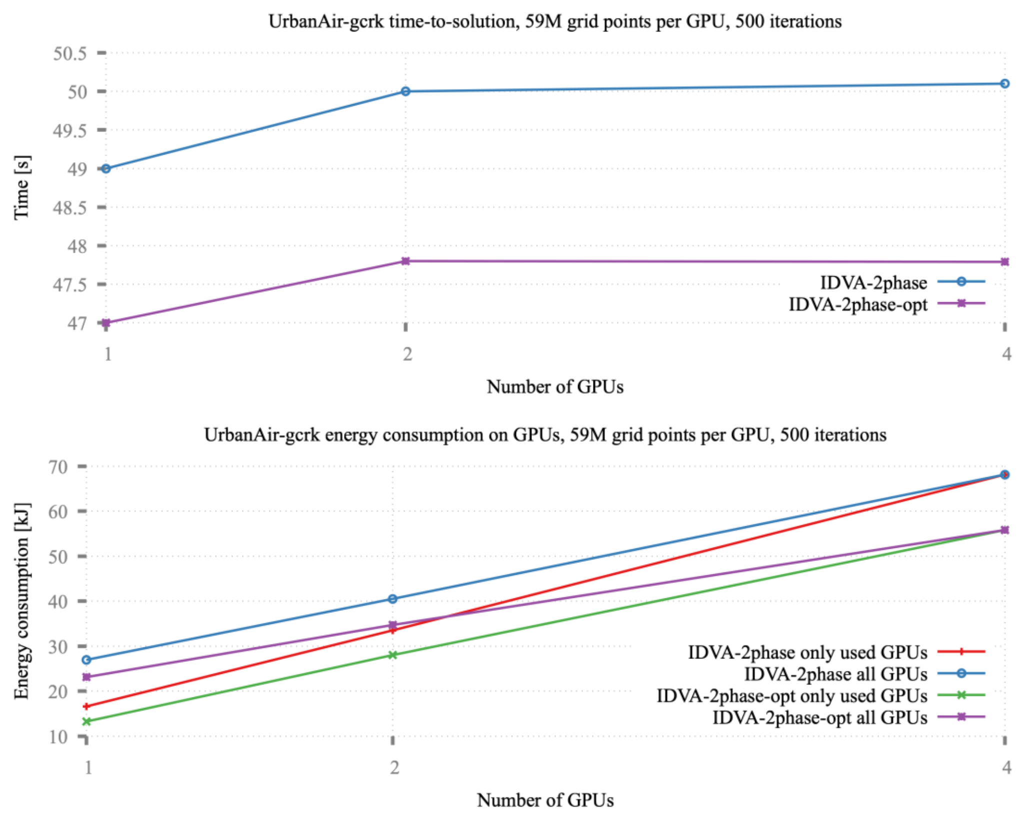
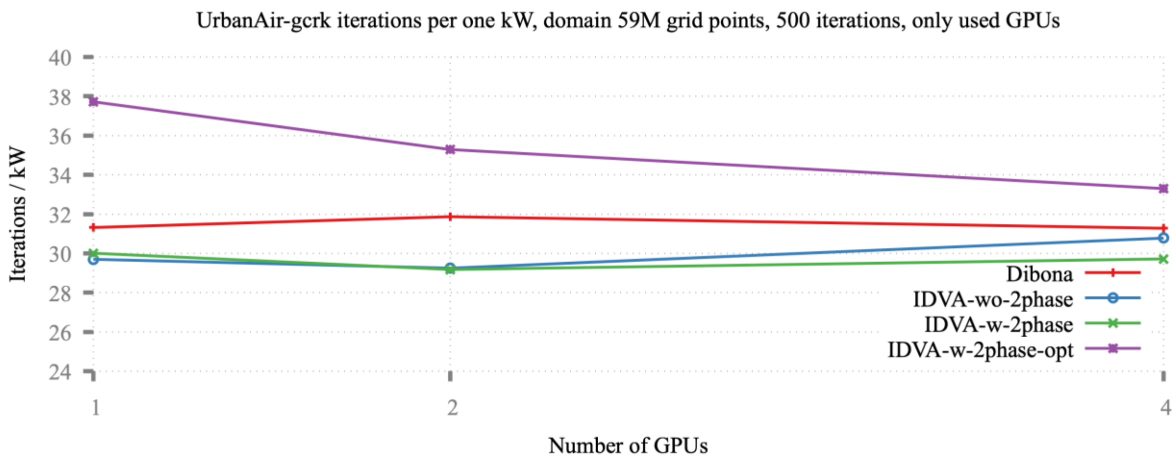


Figure 85: [UrbanAir] Time-to-solution and energy consumption for 59M grid points per GPU

Figure 86 presents the number of iterations per kW for different number of GPUs and systems. When only used GPUs are taken into account, the optimized version running on IDV-A with two phase cooling installed is able to compute the highest number of iterations. The lowest number of iterations per kW is solved with regular version running on IDV-A with or without 2-phase cooling. Similar results are obtained when all GPUs, even unused ones, are taken into account of energy consumption.



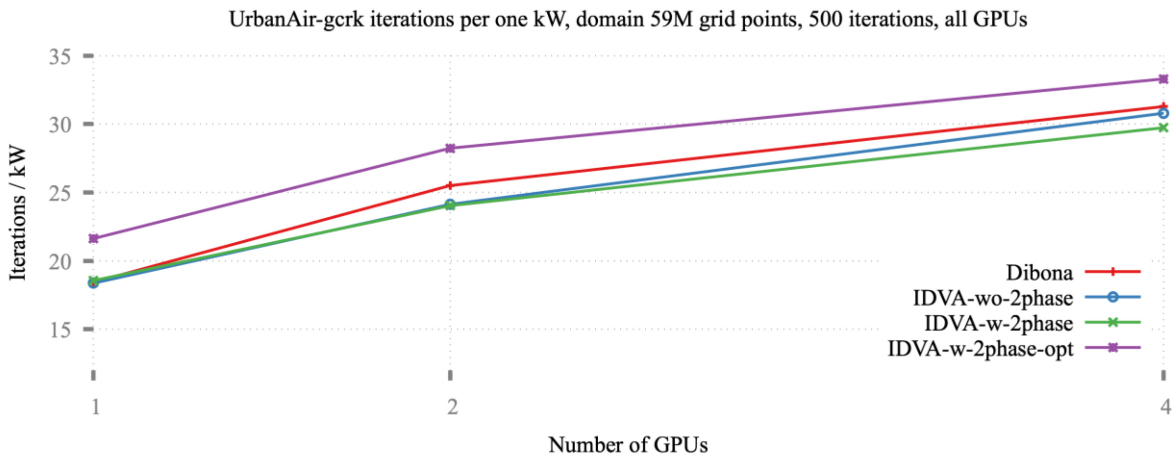


Figure 86: [UrbanAir] Iterations / kW

2.10.4 KPIs

UrbanAir-gcrk KPIs are discussed in the previous subsections, and are here summarized in tabular form:

- For a fixed problem size (strong scalability):
  - Table 30 – Time-to-solution
  - Table 31 – Iterations /s
  - Table 32 – Iterations / kW
  - Table 33 – Iterations / kW (unused GPUs accounted)

- For a fixed problem size per GPU (weak scalability):
  - Table 34 – Time-to-solution
  - Table 35 – Iterations / s
  - Table 36 – Iterations / kW
  - Table 37 – Iterations / kW (unused GPUs accounted)

KPI	Time-to-solution [s]			
GPUs	**Dibona**	**IDVA-wo-2phase**	**IDVA-w-2phase**	**IDVA-w-2phase-opt**
1	53.2	49.5	49.0	47.0
2	28.01	25.9	26.2	25.30
4	14.8	14.0	14.5	13.90

Table 30: [UrbanAir] KPI Time-to-solution (strong)

KPI	Iterations / s			
GPUs	**Dibona**	**IDVA-wo-2phase**	**IDVA-w-2phase**	**IDVA-w-2phase-opt**
1	9.4	10.1	10.2	10.64
2	17.85	19.3	19.08	19.76
4	33.78	35.71	34.48	35.97

Table 31: [UrbanAir] KPI Iterations / s (strong)

KPI	Iterations / kW (used GPUs)			
GPUs	**Dibona**	**IDVA-wo-2phase**	**IDVA-w-2phase**	**IDVA-w-2phase-opt**
1	31.33	29.71	30.01	37.72
2	31.88	29.25	29.18	35.29
4	31.28	30.79	29.73	33.31

Table 32: [UrbanAir] KPI Iterations / kW (strong, used GPUs)

KPI	Iterations / kW (all GPUs)			
GPUs	**Dibona**	**IDVA-wo-2phase**	**IDVA-w-2phase**	**IDVA-w-2phase-opt**
1	18.43	18.36	18.55	21.62
2	25.50	24.13	24.03	28.23
4	31.28	30.79	29.73	33.31

Table 33: [UrbanAir] KPI iterations /kW (all GPUs, strong)

KPI	Time-to-solution [s]	
GPUs	**IDVA-2phase**	**IDVA-2phase-opt**
1	49	47
2	50	47.8
4	50.1	47.79

Table 34: [UrbanAir] KPI time-to-solution (weak)

KPI	Iterations / s	
GPUs	**IDVA-2phase**	**IDVA-2phase-opt**
1	10.2	10.63
2	10	10.46
4	9.98	10.46

Table 35: [UrbanAir] KPI Iterations / s (weak)

KPI	Iterations / kW	
GPUs	**IDVA-2phase**	**IDVA-2phase-opt**
1	30.01	37.72
2	14.92	17.85
4	7.34	8.96

Table 36: [UrbanAir] KPI Iterations / kW (used GPUs, weak)

KPI	Iterations / kW (all GPUs)	
GPUs	**IDVA-2phase**	**IDVA-2phase-opt**
1	18.55	21.62
2	12.34	14.4
4	7.34	8.96

Table 37: [UrbanAir] KPI Iterations / kW (all GPUs, weak)

2.11 FIPLib: FPGA Image Processing Library – ENEA/INFN

2.11.1 FIPLib on single FPGA

In this section we report the results achieved implementing the FPGA Image Processing Library (FIPLib) through the Vitis HLS flow. FIPLib is implemented as a collection of C++ kernels and utilizes the Vitis HLS flow to translate the C++ library's kernels, interconnected through streams, into bitstreams. Currently, FIPLib encompasses nearly 70 functionalities, each designed with a streaming behavior: data are read from the input streams and are written to the output streams. The width of all streams is controlled by a template parameter that determines both the data path width and the available parallelism. The functionalities within FIPLib can be classified as follows:

1. **Stream Management**: Encompasses stream copy, stream split, stream merge operations. These modules can also adapt the stream protocol (internal stream or AXI stream) and the stream width (increase/reduce stream width by factor 2 and 4)
2. **Data Mover**: Serves as the interface between DDR/HBM and streams.
3. **Parameter Initialization**: Establishes weight values for distinct filters like Sobel, Gaussian, and Box filters and for structuring elements for morphological operators.
4. **Pixel-based Operations**: Orchestrates pixel-based operations on input images, yielding output images through operations such as multiplication, subtraction, addition, complementation, logical operations, minimum, maximum, and linear scaling.

5. **Color Space Conversion**: Undertakes pixel-based transformations, converting images from the RGB color space to the YUV color space and vice versa.
6. **Image Masking**: Executes pixel-based operations that derive output images based on pixel values from either of two input images, based on the binary masking image.
7. **Line-based Operations**: Requires complete reading of one or more lines prior to initiating the output of processed images. This category encompasses diverse filters (2D convolution with kernels of dimensions 3x3, 5x5, 7x7), median filtering (3x3, 5x5), morphological operators like dilation and erosion, horizontal mirroring, and 2x up-sampling through bilinear interpolation.
8. **Store and Forward Transformations**: Demands full image read before generating the output image. This includes operations such as histogram equalization, contrast maximization, and vertical mirroring.

All functions share a consistent structure, and they encompass the subsequent parameters:

- Input and Output streams
- Function specific parameters (such as kernel weights in convolutional filters)
- Image size (including the number of rows and columns)
- Number of images to be processed before the function restarts.

These functions, or tasks, can be interconnected via streams and are invoked within a designated DataFlow section. Tasks, within a DataFlow section, are executed concurrently, with synchronization achieved through streaming communications. In Figure 87 we provide an excerpt of a basic image-processing kernel, accompanied by its graphical representation within a process network:

```
#pragma HLS dataflow
loadInput<dt16 >(inMem, s1, ImgSize, NImg);
streamCopy<dt16 >(s1, s2, s3, ImgSize, NImg);
do5x5medianFiltering<dt16 >(s2  ,s4, Rows, Cols, NImg);
diffImages<dt16 >(s3, s4, s5, ImgSize, NImg);
storeOutput<dt16 >(outMem , s5, ImgSize, NImg);
```

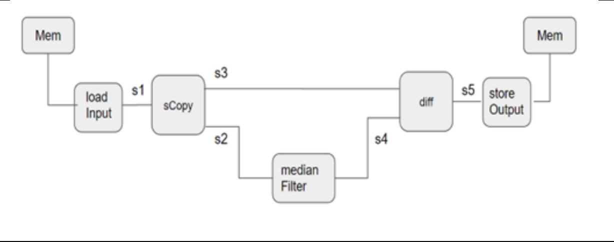


Figure 87:[FIBLib] Codelet and graphics showing a simple image processing kernel

The key point in achieving optimal FPGA processing performance is the maximization of parallelism, both in data access and data processing. Within FIPLib, four distinct types of parallelism are employed:

- **Spatial Parallelism**: This form of parallelism involves the concurrent processing of N image components, optimizing the utilization of parallel resources.
- **Fine-Grain Pipeline Parallelism**: Employed in tasks such as stream management, data movers, and pixel-based operations, this type of parallelism operates at the granularity of an individual pixel component.
- **Medium-Grain Pipeline Parallelism**: Implemented in line-based operations, this type of parallelism is centered around processing an entire line of the image, hence the unit of parallelism is one line of the image
- **Coarse-Grain Pipeline Parallelism**: Observed in store and forward transformations, this parallelism form encompasses the entirety of an image as the unit of parallel processing. It is applicable only when processing a sequence of images.

To demonstrate FIPLib capabilities, let's refer to an algorithm, ALGO1, which draws a pencil-like sketch of a given input color image. ALGO1 uses the following set of kernels, ImgProc, as building block, see Figure 88.

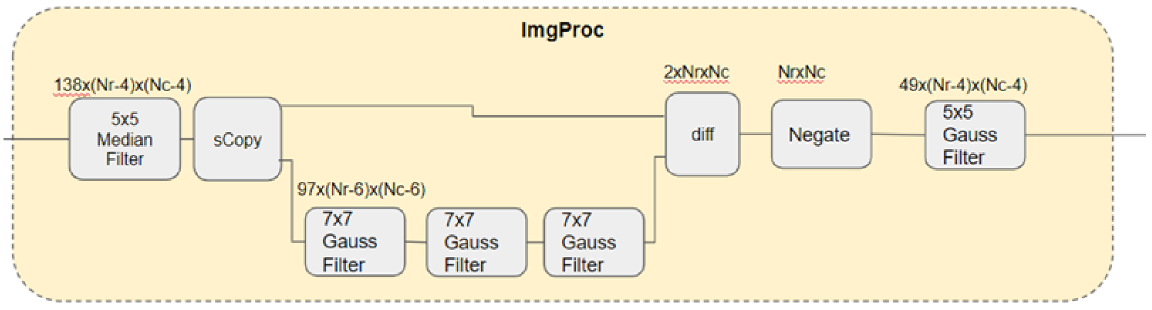


Figure 88: [FIPLib] Basic image processing pipeline used in ALGO1

The ImgProc set of kernels processes images (Rows, Cols) with 8 bits per pixel (8bpp, 256 levels of grey). It cleans the input image through a 5x5 median filter and uses a cascade of three 7x7 gaussian filters to remove the high frequency components of the image; the difference with the cleaned input image allows to extract the high-frequency components (smoothed borders) which are reported as a drawing on a whit paper (Negate kernel) and further smoothed through a final 5x5 Gaussian filter. Thanks to the streaming behavior of the kernels, at the steady state ImgProc can process N pixel components of the input image per clock cycle, being N the width, in bytes, of the streams.

The basic pipeline is instantiated three times to process the R, G, and B channels of a color image. This image is read/written from/to the DDR memory through data movers kernels, with the stream width being twice the width of the processing kernels.

The input image is separated/merged using the splitRGB and mergeRGB kernels. The structure of ALGO1 is reported in Figure 89.

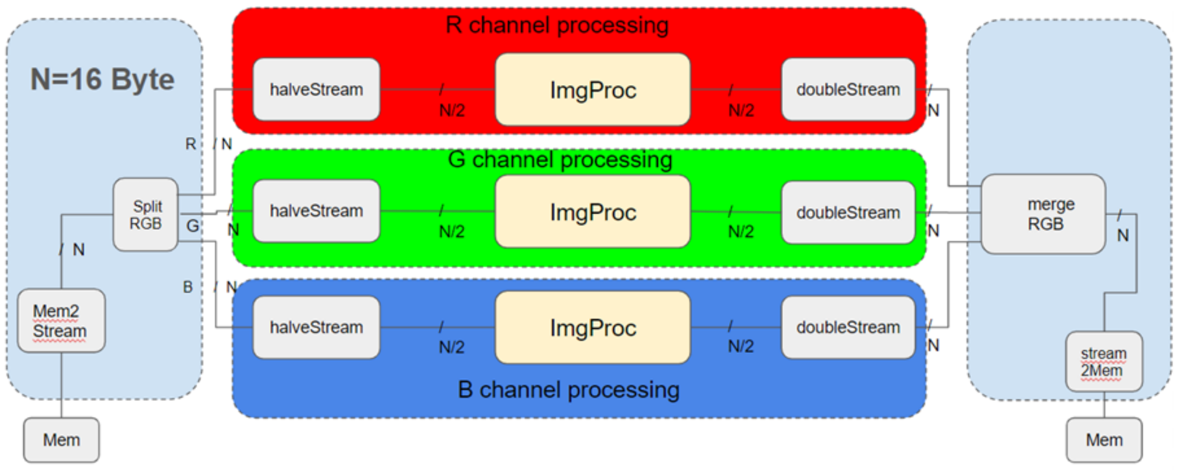


Figure 89: [FIPLib] Kernels in ALGO1

As observed, the size of the streams of the split/merge RGB kernels is twice the size of the in/out streams of the Image Processing Pipelined kernels. This is because data arriving at the R, G, and B channels has 1/3 of the throughput of the memory channels. As a result, new data is presented at the R (or G, or B) channel processing pipeline every 3 clock cycles. Therefore, we halved the size of the streams (and consequently the parallelism) by a factor of two. This approach allows the inner pipeline to process data with an efficiency of 2/3 instead of 1/3.

Counting the number of operations performed by ALGO1 to process a (Rows; Cols) RGB image (refer to the expressions reported in the ImgProc figure), we find that the number of operations to compute ALGO1 is given by

$N_{ALGO1} \approx 1.4 \times 10^3 \times Rows \times Cols$

being operations seized as either 8-bit integer or 16-bit integer (multiply in the Gaussian Filter) or 32-bit (addition in the Gaussian Filter).

After synthesizing the design produced by the HLS flow, we ran ALGO1 to process a sequence of 3000 4096x4096 RGB images. We measured the time from the start of the send of the first image to the FPGA memory to the end of the reception of the last image processed by the FPGA. The time to process all the images is 65.8 s, so one image is processed in $T_{ExeFPGA}$=21.9 ms and the sustained speed is

$S = N_{ALGO1}/T_{ExeFPGA} = 2.35 \times 10^{10}/2{,}19 \times 10^{-2} = 1.07 \times 10^{12}$ Op/s

The resources used by the ALGO1 are reported in Table 38.

	#	%
LUT	121127	10.9%
BRAM	328	19.4%
DSP	4376	48.6%

Table 38: [FIPLib] Resources used by ALGO1

ALGO1 clock frequency is f_{ck}=200 MHz. As the number of clock cycles to process one image is $N_{ck} = T_{Exe} \times f_{ck} = 4.3 \times 10^6$, ALGO1 is sustaining

$N_{ALGO1}/ N_{ck} = 5.5 \times 10^3$ Ops/cycle

To evaluate the advantages of FIPLib, we implemented ALGO1 on a multicore Intel Xeon Haswell CPU using the OpenCV library [see https://opencv.org]. We left to the OpenCV the management of the parallelism and we saw that 9 cores were fully used during the processing. We processed the same sequence of 3000 RGB images used in the FPGA tests. The processing time for the whole sequence is 508.5 s, so the time needed to process one image is $T_{ExeopenCV}$=170 ms and the speed-up achieved through the FPGA implementation w.r.t. openCV is S = 7.8

FPGAs are commonly regarded as computing devices capable of reducing power consumption. Let's assess and quantify this aspect through measurements on FPGA kernels implemented using the FIPLib and compare their energy usage with the corresponding OpenCV implementation on the CPU.

To quantify the power consumption during a computation, let's consider the energy used by the application to be run, being the energy defined as

$$E_A = \int_0^D P_A(t)\, dt$$

where D is the running time of the application, and

$$P_A(t) = P(t) - P_{Idle}$$

the power used by the application at time t, computed as difference between the node instant power and the power absorbed by the node when no user processing is running.

To perform the power measurements, we accessed the Baseboard Management Controllers (BMCs) of the computing node through the Intelligent Platform Management Interface (IPMI): in this way, with a sampling period of 1 s, we can read the instant power absorption of the node. As $P(t)$ is the power erogated by power supply unit, it includes the power used both by the CPU and by the FPGA.
To compute E_A, we first estimate P_{Idle} measuring the instant power absorbed when there are no user processes running, then we take the instant power measurements, along the application run, to compute E_A.
To express the global performance of the execution of an algorithm implementation on a platform, we use the Energy Delay Product (EDP) [12].

$$\text{EDP} = E_A \times D$$

EDP [Js] considers both the energy consumed by the application and its execution time. Using EDP, we can distinguish between two designs that consume the same amount of energy by identifying the faster one. Smaller EDP indicates more efficient design implementation.
Figure 90 reports $P_A(t)$ when running ALGO1 to process the 3000 RGB images on CPU, through OpenCV, and on FPGA, through the FIPLib.

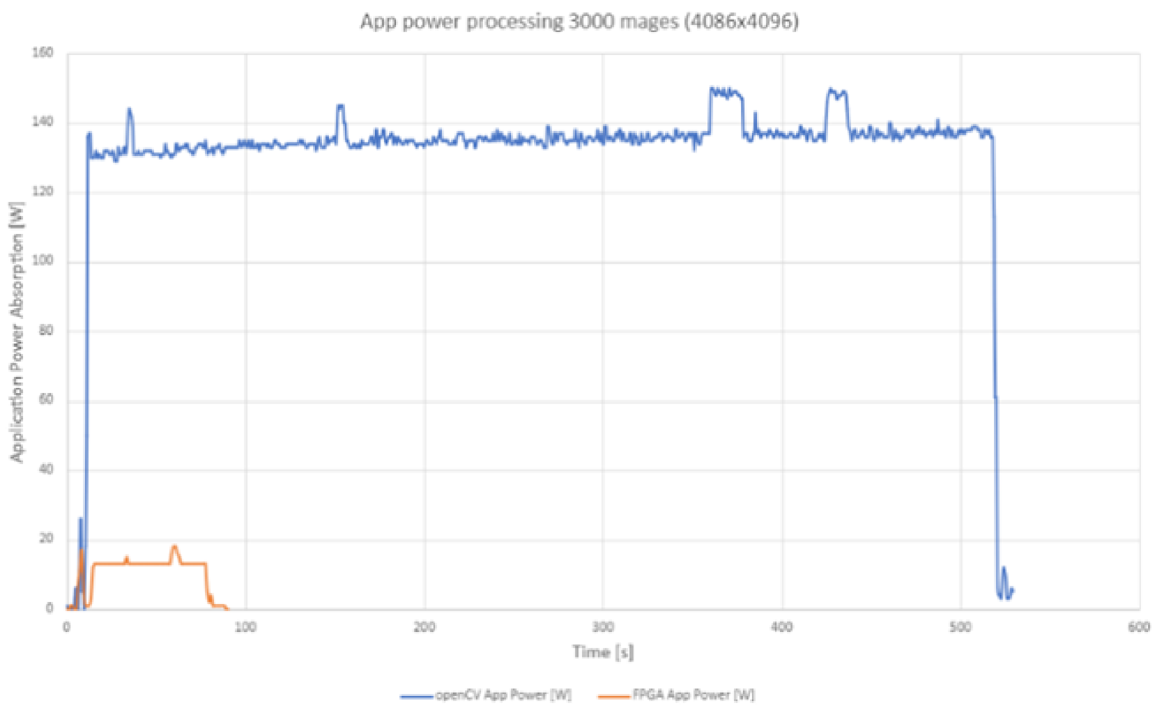


Figure 90: [FIPLib] Power absorption of ALGO1

The values for E_A (i.e. the areas below the two plots), the computing times, and EDP are reported in Table 39 (computing time includes FPGA programming time, filesystem accesses, and all the needed initializations).

	Energy used by ALGO1 run (kJ)	Time used by ALGO1 run (s)	EDP for ALGO1 run [kJ*s]
FPGA / FIPLib	0.92	91	60
CPU / openCV	69.22	530.6	35198

Table 39: [FIPLib] Energy and time used for ALGO1

FPGA implementations outperform CPU implementations in both energy efficiency and speed, using 75 times less energy and 7.8 times less time.

2.11.2 Multi-FPGA Implementation

One of the possible improvements for the application is to perform image processing by working on streams of pixel of 256 bit width. However, this possible update translates in a huge requirement in terms of resource to implement the RGB kernels described in the previous section (in particular, since each ImgProc hw function requires ~32% of the available DSPs of the board). This made it impossible to implement the total application on a single FPGA. To cope with this problem, we chose to deploy FIPLib on a multi-FPGA setup exploiting the APEIRON framework: in this way, we were able to split the overall image processing by implementing a single RGB kernel on each node, each of them dedicated to a single color stream processing. Implementing FIPLib HLS kernels as APEIRON tasks means to change the interface of each of them (to cope with the standard required by the framework to compile the entire project. An example of this type of change is reported in Listing 1.

SINGLE FPGA FPLib IMPLEMENTATION

```
extern "C" {
void ImgProc(
          hls::stream<io_stream_16B>  &s_in,
          hls::stream<io_stream_16B>  &s_out,
          unsigned int ImgSize,
          unsigned int NbImages,
          unsigned short int ImgRows,
          unsigned short int ImgCols,
          unsigned int channel)
{
...
}
}
```

MULTI-FPGA FPLib IMPLEMENTATION (APEIRON)

```
#include "ape_hls/hapecom.hpp"

extern "C" {
void ImgProc(
          message_stream_t
message_data_in[N_INPUT_CHANNELS],
          message_stream_t
message_data_out[N_OUTPUT_CHANNELS],
          unsigned int ImgSize,
          unsigned int NbImages,
          unsigned short int ImgRows,
          unsigned short int ImgCols,
          unsigned int channel_id)
{
...
}
}
```

Listing 1: [FIPLib] Single vs multi FPGA

Another requirement for the implementation via APEIRON was to substitute the basic Vitis HLS stream connection approach (stream.write, stream.read) with the HAPECOM APIs send() and receive() functions in order to allow the communication between them by exchanging Apelink packets through the network. A snippet of the code is reported in Listing 2.

SINGLE FPGA FPLib IMPLEMENTATION	MULTI-FPGA FPLib IMPLEMENTATION (APEIRON)

while (NbWordToTransfer > BUFFER_SIZE)
{
    if (phase){
            buffer2Stream(outStream, Buff1, BUFFER_SIZE);
            stream2Buffer(inStream, Buff2, BUFFER_SIZE);
    }
    else{
            buffer2Stream(outStream, Buff2, BUFFER_SIZE);
            stream2Buffer(inStream, Buff1, BUFFER_SIZE);
    }
     phase = !phase;
     NbWordToTransfer -= BUFFER_SIZE;
}
 	
void buffer2Stream(hls::stream<io_stream_16B>& outStream, dt16 Buff[BUFFER_SIZE], unsigned int size)
{
#pragma HLS inline off
        io_stream_16B tmp;
       tmp.keep = 0xFFFF;
       tmp.last = false;
       // copy Buff to stream
        for (unsigned int i = 0; i<size; i++){
#pragma HLS pipeline
tmp.data = Buff[i];
outStream.write(tmp);
        }
}
	#include "ape_hls/hapecom.hpp"
	
	while (NbWordToTransfer > BUFFER_SIZE)
	{
	    if (phase){
	            send(Buff1, BUFFER_SIZE*sizeof(word_t), coord,
	task_id, ch_id, message_data_out);
	             stream2Buffer(inStream, Buff2, BUFFER_SIZE);
	    }
	    else{
	            send(Buff2, BUFFER_SIZE*sizeof(word_t), coord,
	task_id, ch_id, message_data_out);
	            stream2Buffer(inStream, Buff1, BUFFER_SIZE);
	    }
	     phase = !phase;
	     NbWordToTransfer -= BUFFER_SIZE;
	}

Listing 2: [FIPLib] Single vs multi FPGA implementation

The system was composed by 4 interconnected Xilinx® Alveo U200 (installed in the INFN Roma1 APE Lab) in a ring topology. The bitstream flashed on each board implemented a project depicted in Figure 91 and composed by 3 different HLS kernels:

- *RGB2Mem:* connected to INFN Communication IP intranode port 0, it is built combining the overcited *mem2stream* and *splitRGB* kernels
- *Mem2RGB:* connected to INFN Communication IP intranode port 0, it is built combining the overcited stream*2mem* and *mergeRGB* kernels

- *ImgProc:* connected to INFN Communication IP intranode port 1, its functionality is described in the previous setup.

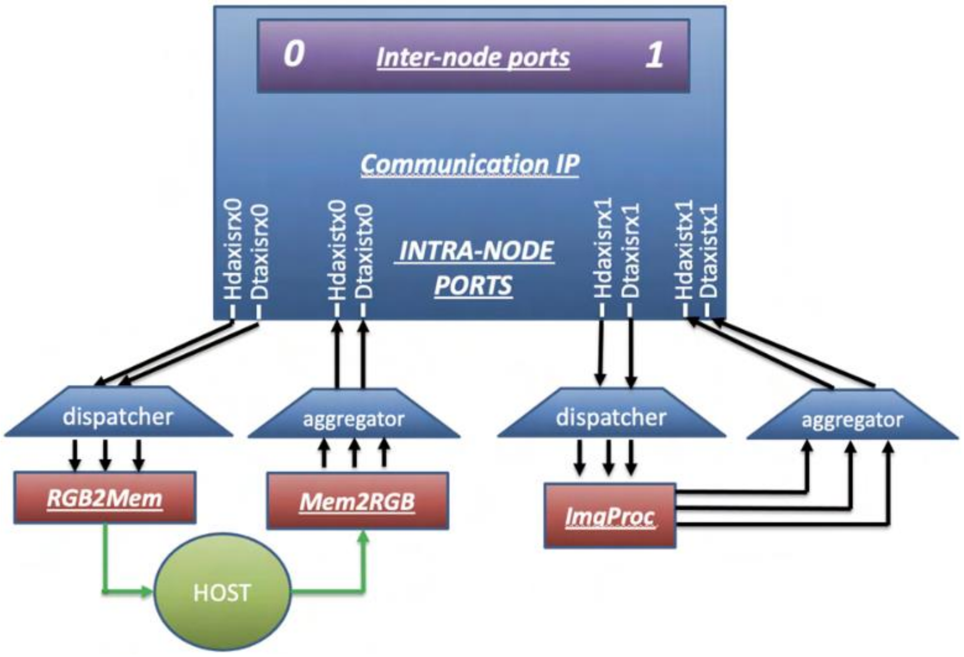


Figure 91: [FIPLib] APEIRON project for single FPGA bistream

In the final setup to be deployed on the four-FPGA system, each board (and so each hardware project flashed) is enabled to work on a specific task. In fact, the FPGA 0 is used to load the RGB images from the memory, to split them into color streams of data and to send them through the network in which the FPGA 1, 2 and 3 are used to process respectively a single color stream of data and to send back results to the FPGA 0 memory.
The FIPLib multi-FPGA deployment and execution schemes are depicted in Figure 92, while a detail on which kind of processing HLS kernels (previously descripted) are used on each board is represented in Figure 93.

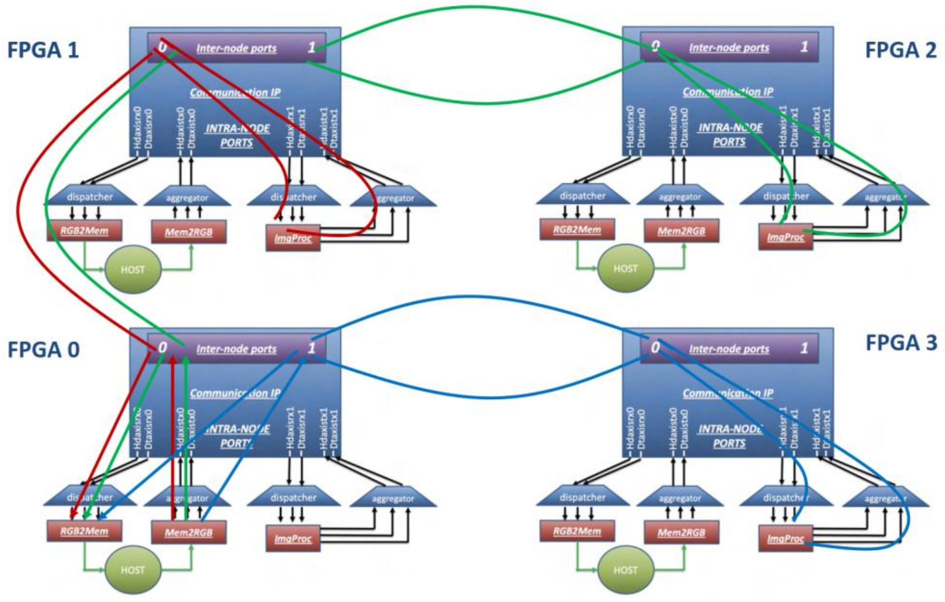


Figure 92: [FIPLib] multi-FPGA deployment and execution scheme

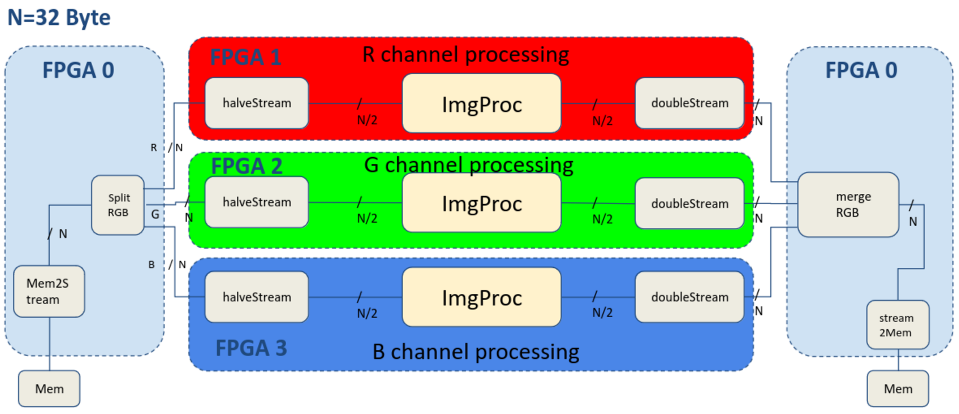


Figure 93: [FIPLib] Kernel displacement in multiple FPGA setup. Internal datapath set at 32 bytes

To test the application performances, we choose to FIPLib on images of difference sizes, measuring the execution time and the power consumption in each of the cases.
In detail, FIPLib execution time and power consumption has been measured exploiting the XRT runtime library, which allows the user to measure FPGA internal voltages and currents values during the application run and to insert time checkpoints to analyze the timeline of running HLS kernels.

The performances result for processing images of sizes 512x512 and 4096x406 are respectively depicted in Figure 94 and Figure 95 and reported in Table 40.

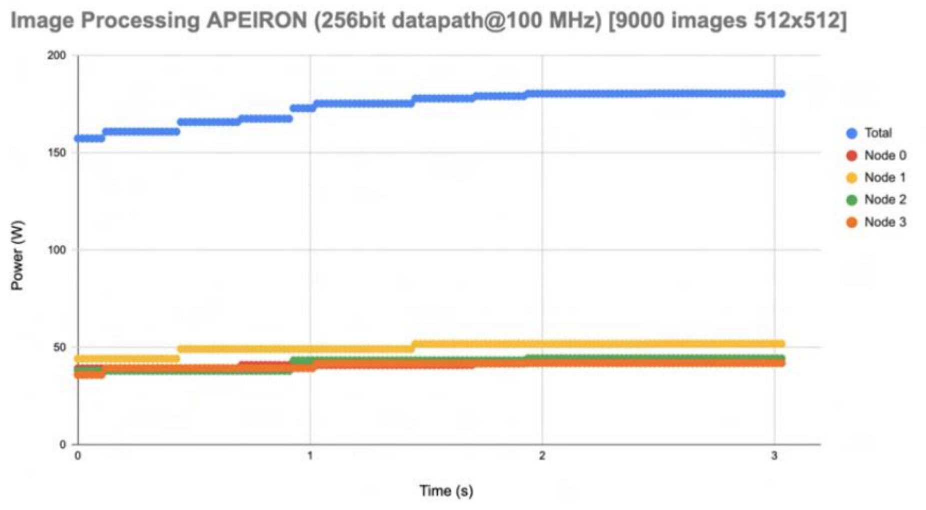


Figure 94: [FPLIB] Multi FPGA APEIRON Image Processing on 512x512 images execution power profile

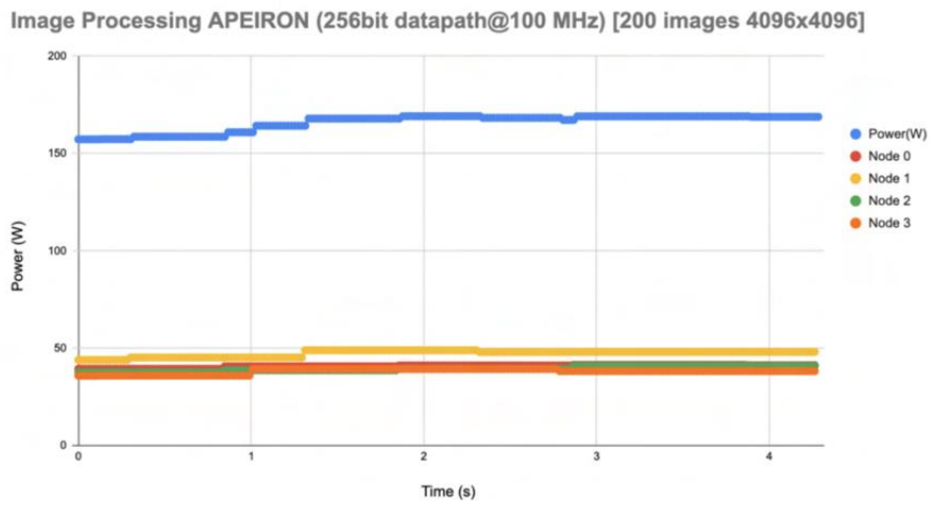


Figure 95: [FPLIB] Multi FPGA APEIRON Image Processing on 4096x4096 images execution power profile

	Image Size	
	512x512	4096x4096
Processing Time	3.05 s	4.28 s
Throughput	2950.62 fps	46.70 fps
Energy consumption	0.53 kJ	0.71 kJ
Processed Images per Joule	16.98 fpJ	0.28 fpJ

Table 40: [FIPLib] Multi-FPGA APEIRON Implementation performances

2.11.3 Single- and Multi-FPGA implementation comparison

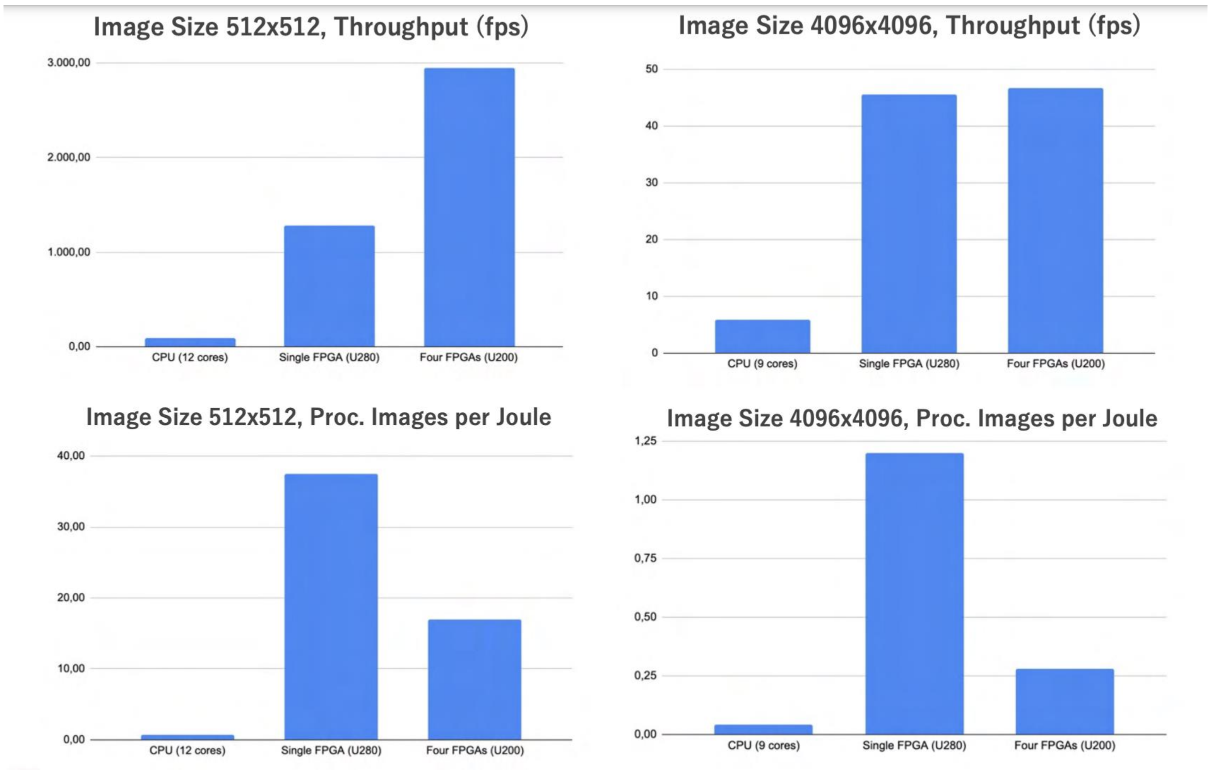


Figure 96: [FIPLib] CPU, Single- and Multi-FPGA implementation performances comparison

In Figure 96, performances obtained from the previous described FIPLib implementations (CPU, Single-FPGA, Multi-FPGA) are compared focusing on throughput and processing images per joule values obtained from the processing of different sizes of images.
In terms of throughput, it can be seen a large improvement while working on 512x512 images on a multi-FPGA setup: this is coherent since the possibility of increase the system datapath to 32 byte wrt to the limit of 16 bytes in the single-FPGA system. A slightly improvement can be seen also while working on 4096x4096 images: in this case the throughput is limited from the bandwidth of the INFN Communication IP used in the APERION framework (since the size of packets to be sent on the network scales with the numbers of columns of the image).
In terms of energy, the scaling in number of used FPGA boards is coupled to a huge loss in processing images per joule (higher loss in energy when lower improvement in throughput).

2.12 NBody - BSC

This section shows the results of the N-Body simulation application. This application allows evaluation and comparison of different use cases of the IDV-E computing node. It also provides a way to verify the successfulness of the different software stack integrations developed in the project.
This section presents results when running the application in a single FPGA, multiple FPGAs without FPGA-to-FPGA communication, multiple FPGA with direct FPGA-to-FPGA

communication using APEIRON communication framework and finally, multi-node multi-FPGA using OMPIF over ethernet.
N-body simulation computes the interaction of a set of particles due to gravitational forces. The input is a set of particles with initial positions, velocities, and masses. Both positions and velocities are represented by 3-dimensional vectors. Data is represented with single precision. During the simulation, two steps are repeated iteratively. The first one is the computation and accumulation of forces that each particle exerts over all other ones. This part is the most computationally expensive, because the number of forces grows proportional to n^2 where n is the number of particles. The second one is the update of the particle velocity and position for a given time step. Therefore, most of the resources are dedicated to the force computation step.

2.12.1 OmpSs@FPGA on IDV-E single FPGA

This implementation uses one of the two available FPGA devices, it's used as a baseline and by a means to evaluate thermal and power profile of the IDV-E cooling system described in deliverable D3.2.
The design is created using the OmpSs@FPGA toolchain [13] which is described in deliverable D4.6. Figure 97 shows a high-level diagram of the resulting design.

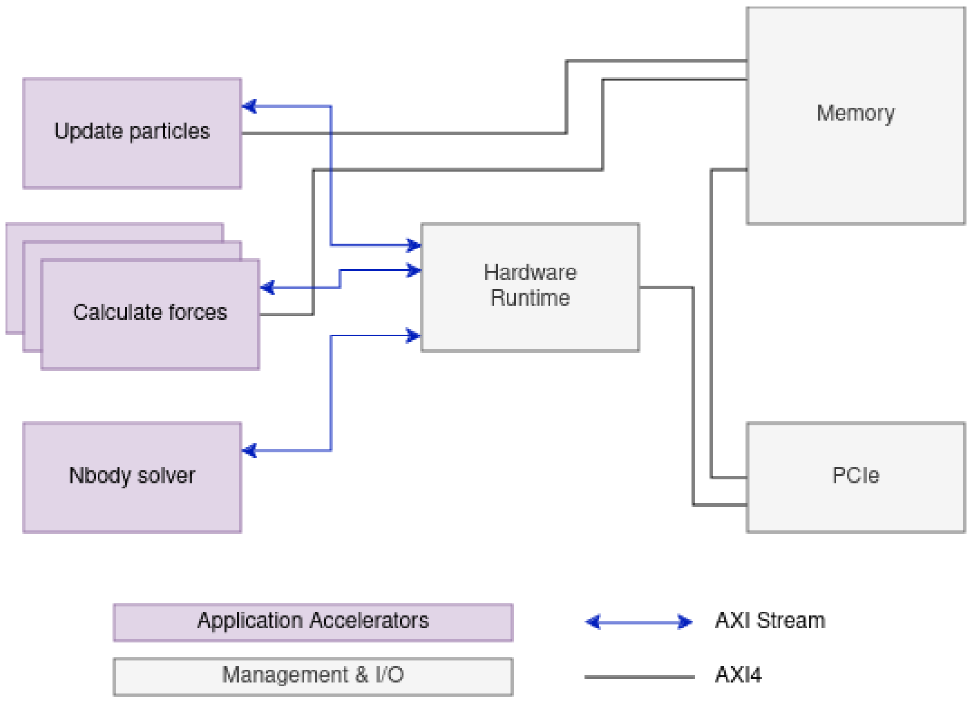


Figure 97: [NBody] Single node n-body FPGA design

Each block represents an IP core or module, and lines represent data buses. Purple boxes represent each of the application accelerators in the design. The stacked boxes represent multiple instances of the same accelerator. Gray boxes represent management modules and I/O infrastructure.
The n-body design implemented contains one instance of the *Nbody solver* and *update particles* kernels and 8 instances of the *calculate forces* accelerators. Each of the *calculate forces* kernels

calculates forces for 16 particles in parallel. This distribution of resources is done because force calculation is the most compute intensive part of the application.
Table 41 shows the amount of resources used by this design.

Resource	Used	Available	Used %
LUT	600625	1303680	46.071507
LUTRAM	99094	600960	16.489285
FF	800548	2607360	30.703392
BRAM	614	2016	30.456348
URAM	8	960	0.8333334
DSP	5131	9024	56.859486
IO	6	624	0.9615385
GT	16	24	66.66667
BUFG	12	1008	1.1904762
MMCM	1	12	8.333334
PCIe	1	6	16.666668

Table 41: [NBody] Resource usage of the single-node n-body for the IDV-E alveo U280

In this implementation, the host copies data to FPGA memory and then submits a single FPGA task that will solve the problem for a given number of particles and timesteps. This is shown in Figure 98.

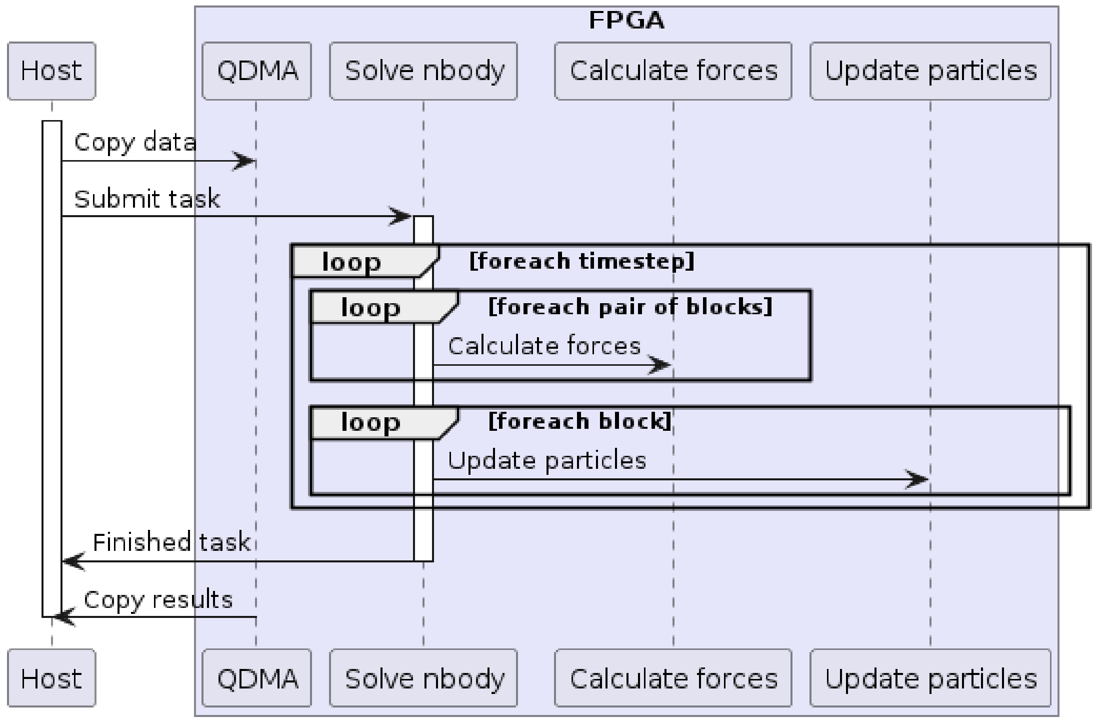


Figure 98: [NBody] Diagram sequence of the n-body computation for single FPGA

The host first copies data to the FPGA memory by submitting a DMA operation to the QDMA module, which implements the PCIe communication between the host and the FPGA. Then, a single *solve nbody* task is submitted. This task then communicates with the hardware runtime to create and submit tasks to the *calculate forces* and *update particles* accelerators in order to solve the n-body problem for a given number of particles and time steps. Finally, the host copies the results back to main memory after the *solve nbody* task is finished.

Single FPGA implementation reaches a performance of 37.43 Gpps (Giga pairs per second) while consuming 94.84W. This yields a power efficiency of 0.395 Gpairs/Watt.
More details as well as comparison with other platforms are described in deliverables D1.4 and D4.6.

Regarding thermal performance, IDV-E shows better behaviour when comparing to other platforms. We compared against actively air-cooled Alveo U200, a passively cooled Alveo U55c and the U280 using in quattro two-phase cooling technology from IDV-E as described in deliverable D3.2.
Regarding memory configurations, Alveo U200 uses 64GB DDR memory, Alveo U55c integrates 16GB HBM memory instead of DDR and IDV-E Alveo U280 has 8GB HBM and 32GB DDR, however, in our tests, only HBM is used.
All three devices make use the same design, using roughly the same number of resources. Therefore, power draw from design logic is expected not to change across the devices as they are the same device family (AMD Virtex UltraScale+) using the same manufacturing process.

Figure 99, Figure 100, Figure 101 and Figure 102 show temperature and power usage for during an n-body execution of the same problem size in U200, U55, and IDV-E's U280 air cooled and 2 phase cooled devices respectively.

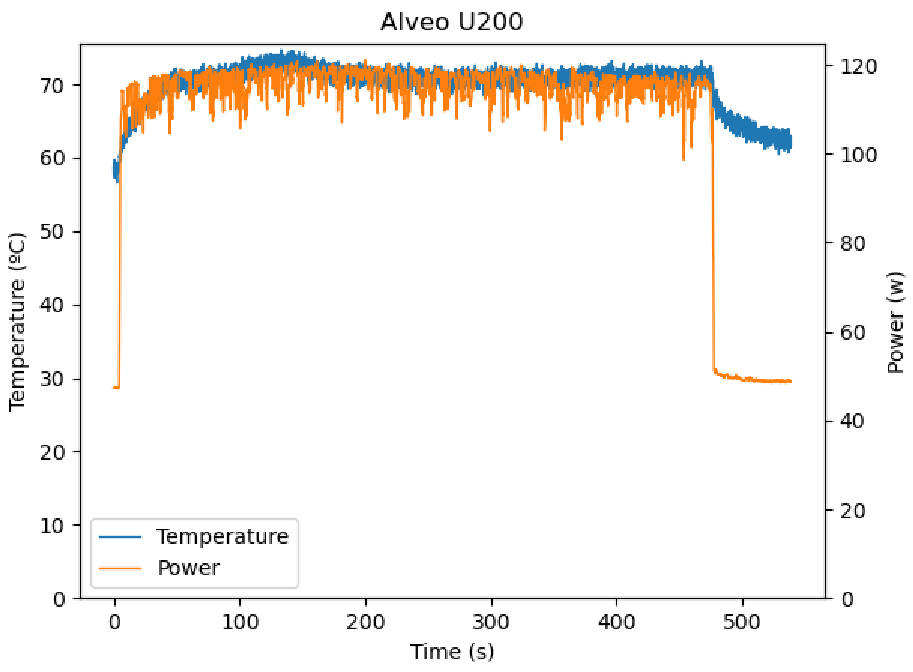


Figure 99: [NBody] Temperature and power for an actively air-cooled Alveo U200

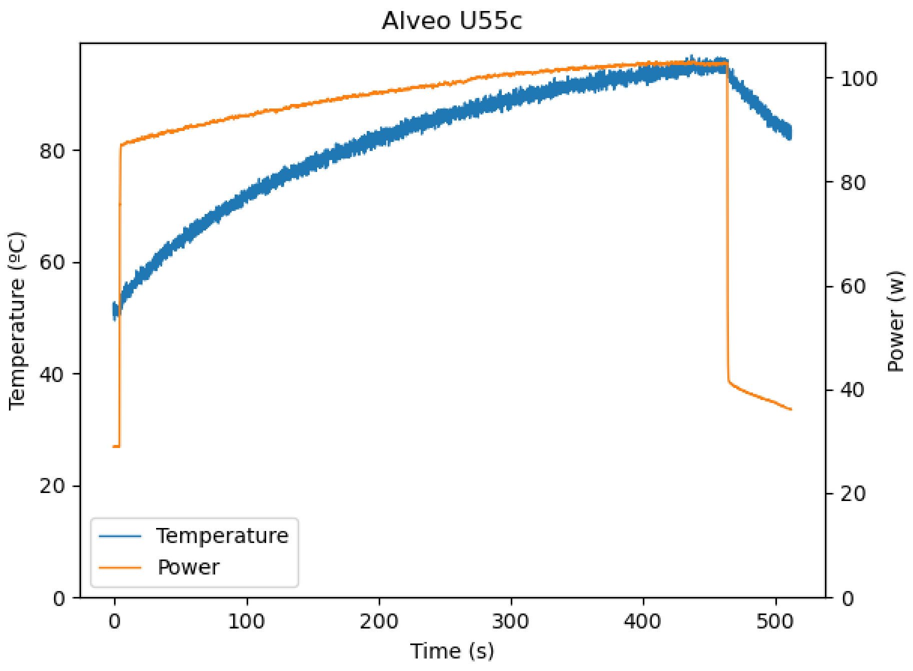


Figure 100: [NBody] Temperature and power for a passively air-cooled Alveo U55c

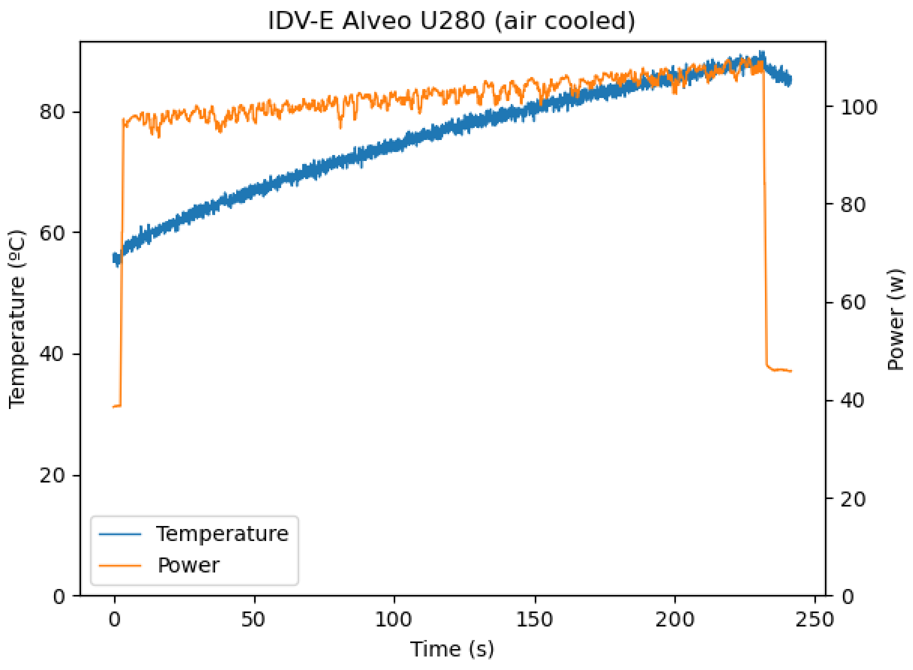


Figure 101: [NBody] Temperature and power air cooled IDV-E's Alveo U280

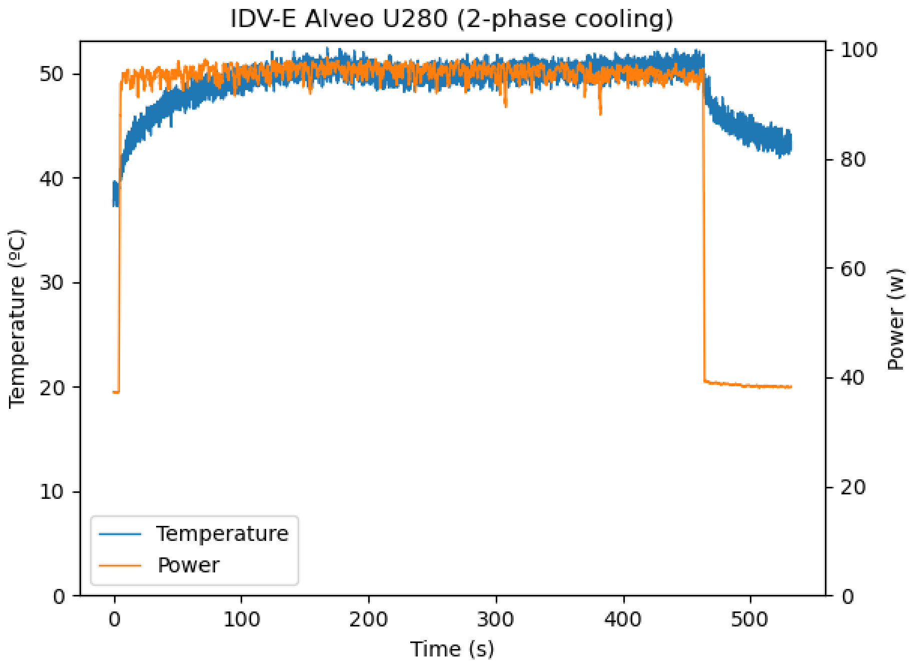


Figure 102: [NBody] Temperature and power 2-phase cooled IDV-E's Alveo U280

Those figures show many differences regarding temperature highlighting differences in the cooling solutions used in each case.
The temperature for the Alveo U200 seems to remain stable throughout the execution. This is the expected outcome as the fan controller should adjust fan speed depending on the thermal load. This can be seen in Figure 99 at around 150s into the execution, the temperature slightly drops as the fan speed increases. Also, idle temperature is higher than the other two platforms as the fan will run at less speed at lower thermal loads.
Alveo U55c shows lower idle temperature. This is due to the case airflow being relatively high disregarding FPGA's thermal load, which is the expected behavior in a passive cooling solution. However, when computation starts, the amount of dissipated power increases, but the airflow does not significantly increase, causing the temperature to increase throughout execution reaching 96ºC in this execution. Insufficient airflow when thermal demand is high could lead to thermal protection shutting down the device to prevent catastrophic failure.
Finally, IDV-E cooling system shows bigger thermal mass as temperature increases at a lower rate than other devices at the start of the execution. A larger thermal mass allows to absorb peaks in thermal load. Temperature drops slightly at around 160 seconds; this suggests that the cooling system is reacting to the change in temperature as was the case for active cooler in the Alveo U200.

Those charts show also large differences in power usage across the evaluated devices and temperatures.
On Alveo U200, average power usage is 110W, which is higher than the 95W that the U280 uses. This difference can be attributed to the different memory used. This board uses 64GB DDR memory instead of the 16GB DDR + 8GB HBM of the U280. Also, active cooling has an impact, since the integrated fan has a non-negligible power draw that also is included in measurements.

On Alveo U55c, even though power draw is at the start of the execution is 80W, this quickly increases due to the temperature change in the FPGA chip. This is a known effect in MOSFET-based integrated circuits [14]. At the end of the execution power draw is just over 102W. On longer executions, power seems to stabilize at around 103 W.
On the IDV-E, when using air cooling, behavior is similar to the U55c. However, cooling performance is worse. In fact, executions are shorter because on longer executions, overtemperature protection is tripped to avoid catastrophic failure. This can be seen as the x axis on Figure 55 shows a shorter span of time. This is caused by the airflow in the node being insufficient to properly cool the boards. On longer executions, the power draw reaches around 115W just before the device powers down itself. Reported power consumption reported is slightly lower as we run shorter executions in order not to get so close to the thermal limit.
In the 2-phase cooled IDV-E power usage remains stable throughout the execution, staying close to the 94W average. On top of running cooler overall, this cooling method allows us to run workloads for extended periods of time without triggering thermal protection.
All in all, power efficiency of the FPGA in 2 phase cooled IDV-E is highest of the devices evaluated, as shown on Table 42. Cooling solution plays an important role in power efficiency, as the device running hotter results in higher power draw. Even in the case of passive solutions the influence of temperature on power consumption can negate power savings of the active components. Furthermore, temperatures can reach dangerous levels in particular workloads if server chassis air flow is not adjusted accordingly.

Device	Performance (Gpps)	Power (W)	Energy efficiency (Gpps/W)
U200	36.4214	112	0.3265
U55c	37.4103	103	0.3629
U280 IDV-E (air)	37.4343	107	0.3487
U280 IDV-E (2-phase)	37.4348	94.8	0.3947

Table 42: [NBody] Comparison of performance, power and power efficiency across different devices

These power measurements only consider the FPGA device's power. In all cases except for the Alveo U200, cooling is passive and therefore power used by the cooling infrastructure is not included in measurements. Table 43 shows power consumption of the full node including cooling infrastructure for the air-cooled and 2-phase-cooled versions of the IDV-E when using 1 and 2 FPGAs.

Power (W)	1 FPGA	2 FPGA
Air cooling	354.3846154	424.8186813
2 phase cooling	384.1043956	447.6978022

Table 43: [NBody] Comparison of power consumption on air cooled and 2-phase cooled IDV-E

As table shows, when considering the power consumed by the cooling infrastructure, it consumes more power than the air-cooled alternative for this configuration. However, the difference in consumed power when using one or two FPGAs is lower in the two-phase cooling system (+16.5%) than in the air-cooled version (+19.8%). Figure 103 shows a comparison between both versions of the IDV-E. For the cooling version, a breakdown of the used power between the computing node and the cooling system. For the air-cooled version, measuring power consumed by cooling fans is not possible as only full node power usage is reported.

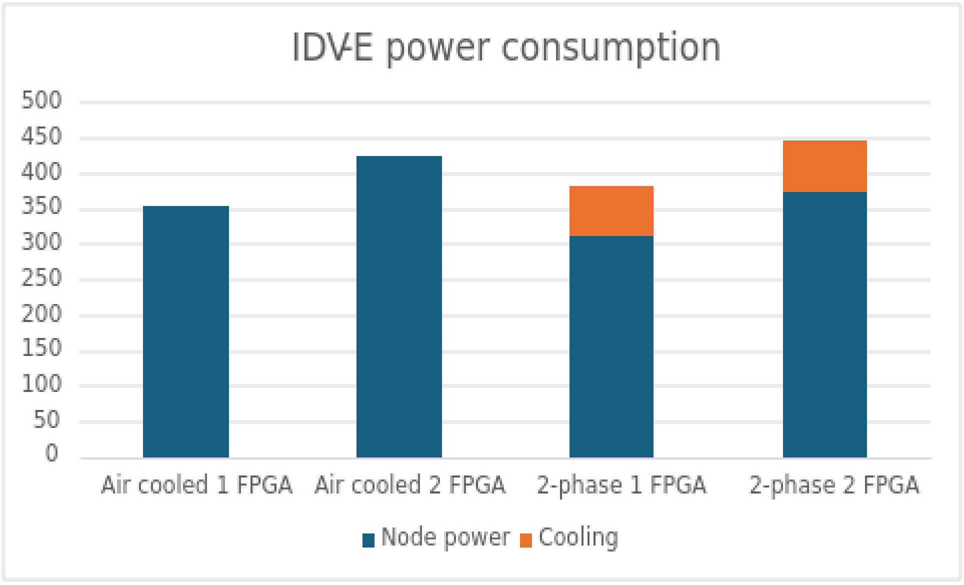


Figure 103: [NBody] Sequence diagram of multi-fpga n-body execution

Power consumption of the 2-phase cooling system remained constant at 73W throughout all experiments independently of the power used by the IDV-E node.
Therefore, if the system used more FPGAs and the trend continues, a 2-phase cooled IDV-E node with more FPGAs would be more efficient than the air-cooled alternative.
This trend is shown in Figure 104.

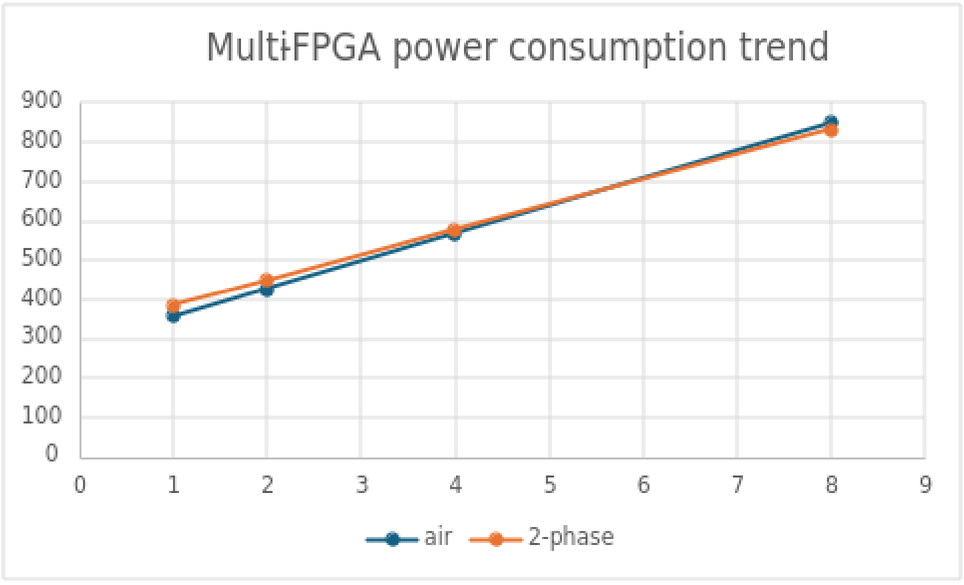


Figure 104: [NBody] Sequence diagram of multi-fpga n-body execution

This figure shows the power consumption trend for air-cooled and 2-phase-cooled IDV-E for different number of FPGAs.
For nodes with 8 FPGAs as the ones used in MEEP project [15], two-phase cooling would be more efficient than air-cooling the FPGAs. Also, in our experiments, CPUs are only used for

FPGA task submission, no workload is being executed in the CPU, leaving almost all the cores in idle state. If workloads actively use CPUs, we expect 2-phase cooling to be even more efficient as cooling load increases.
It is also worth noting that the current air-cooled configuration is insufficient and cannot be used for workloads that run in the FPGAs for an extended time.

2.12.2 OmpSs@FPGA on IDV-E multiple FPGA

In this use case, we use both FPGAs in the IDV-E to run the n-body simulation. In this case, the host processor manages execution across the FPGAs without relying in the FPGAs directly communicating between them.
To distribute work among multiple FPGAs, it's not possible to submit a single task that computes the full n-body problem. The host process must create tasks for each particle block and move data from one FPGA to another. This workflow is shown in Figure 105.

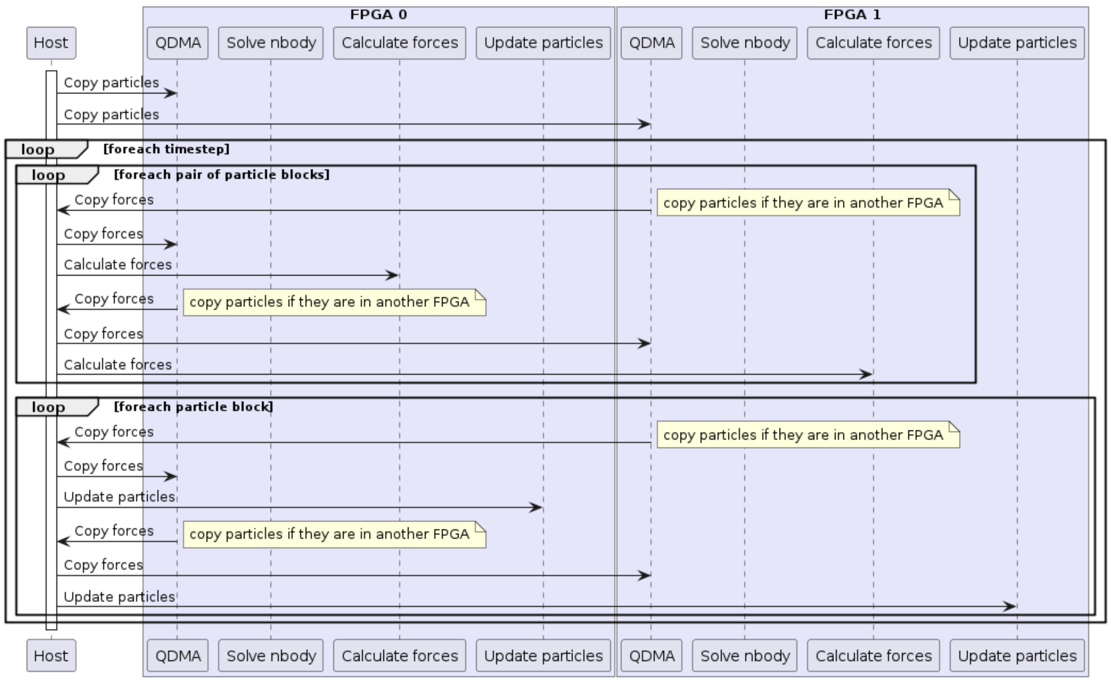


Figure 105: [NBody] Sequence diagram of multi-fpga n-body execution

The sequence diagram in Figure 105 clearly shows that the amount of communication between the host system and the FPGA is much larger than the single device implementation described in Figure 97. This is true for task execution commands since the host must send an individual command for each task to process every block of particles. Also, the amount of data copies quickly grows. Data that needs to be moved from one device to another needs to be copied from the source device memory to host main memory, and then copied again to the destination device memory. If P2P transfers were supported, data could be copied from one device to another without going through the host. This can decrease the total amount of data movements, but keeping track of where each individual block of particles is stored in a given point in time has a non-negligible overhead that will limit performance as the problem size and number of devices grow.

Another disadvantage of this approach is that task scheduling quickly becomes a bottleneck. The host may not be fast enough creating tasks, checking that their dependences are ready and sending them to the appropriate device to keep accelerators busy.

All in all, there's a performance degradation when moving to a 2-device implementation. This is shown in Figure 106.

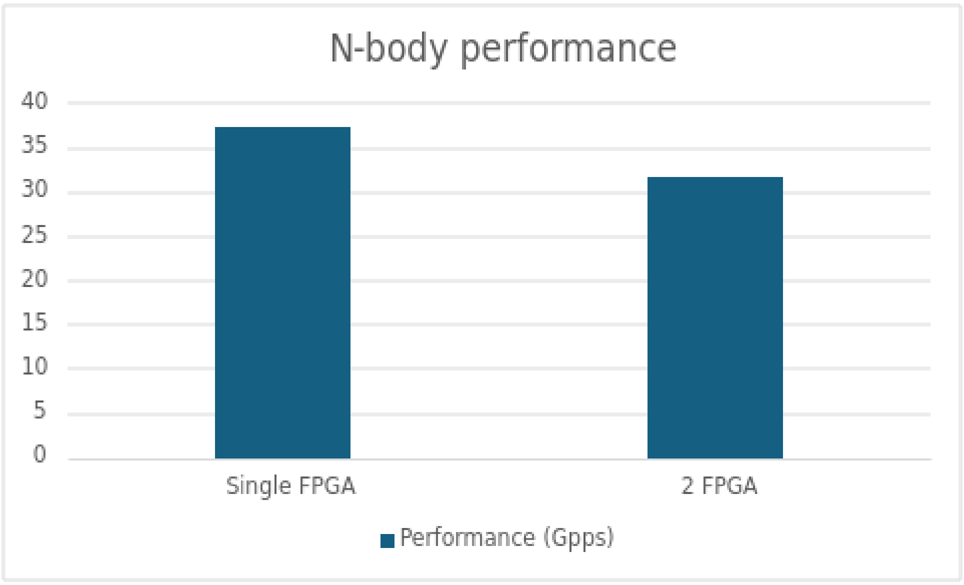


Figure 106: [NBody] Performance comparison between using 1 and 2 FPGA

This chart shows performance in Giga pairs per second (Gpps) for single and multi FPGA implementation. The design used for the multi-FPGA is the same as in the single node implementation shown in Figure 97, but the *solve nbody* accelerator is not used. Tasks are submitted directly submitted to *calculate forces* and *update particles* accelerators.

Even though this approach may work for problems that can be partitioned in larger blocks that do not need communication between them, the all-to-all pattern of the n-body simulation makes it unfeasible to efficiently implement this application following this approach.

2.12.3 OmpSs@FPGA + APEIRON

To overcome issues regarding data copies and task scheduling of the host directly managed multiple devices, we implemented direct FPGA-to-FPGA data transfers. Communications between FPGA devices are implemented using APEIRON framework from INFN (described in D2.9). We use the apeiron API to send and receive data from the application accelerated kernels to the switch. The switch then routes packets to the destination FPGA in the network. Figure 107 shows a diagram representing the design of the of the n-body using apeiron to implement inter-FPGA communications.

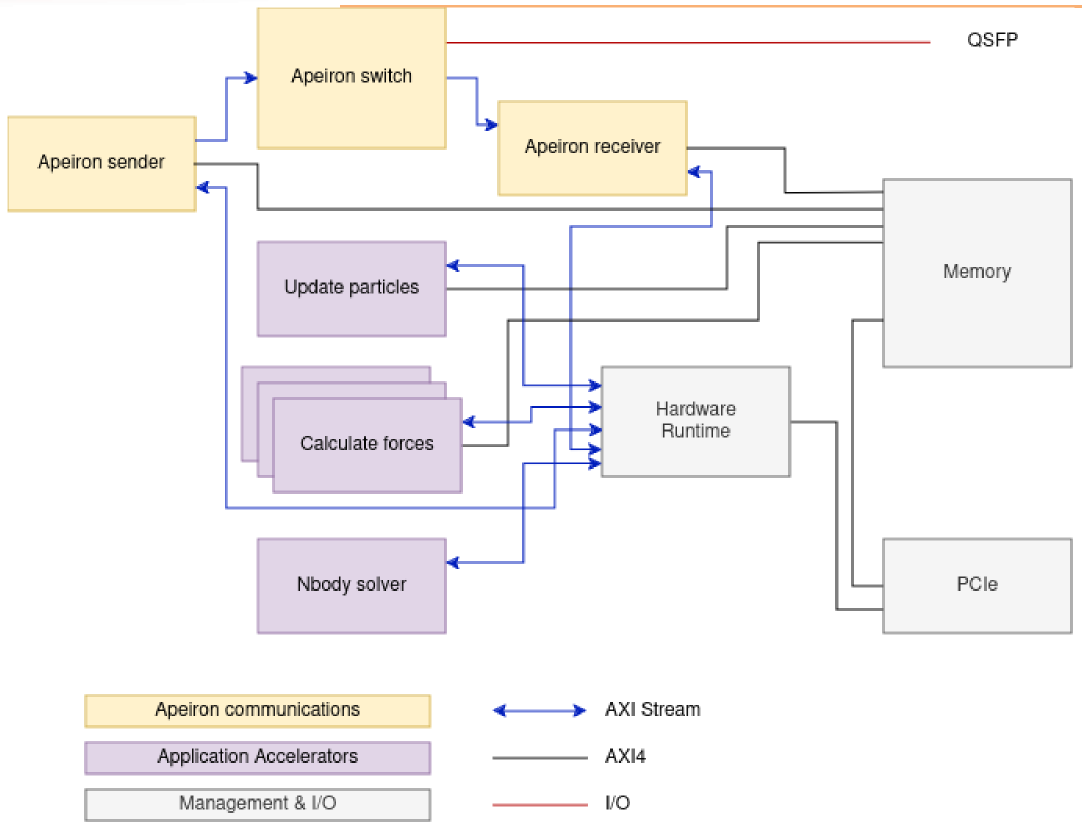


Figure 107: [NBody] body design diagram showing ompss accelerators and apeiron communication modules

Each of the blocks shown in the figure represents different IP cores in the design. The diagram shows application accelerators (purple blocks) connected to the hardware, which schedules tasks and to device memory. Hardware runtime sends commands to accelerators using an AXI stream interface. Accelerators read and write data to memory using AXI-4 interfaces. Yellow blocks are the apeiron communication modules. Apeiron sender and Apeiron receiver modules are connected like other application accelerators.

When an accelerator needs to send or receive data, a command is sent to the apeiron sender or receiver. Then it reads data from memory, builds apeiron packets and sends them to the switch through a stream interface. Then the switch sends the data though the appropriate QSFP port on the board. The receiver works in the opposite direction, receives data from the switch and writes it to memory. This process is shown along with all n-body execution flow in Figure 108.

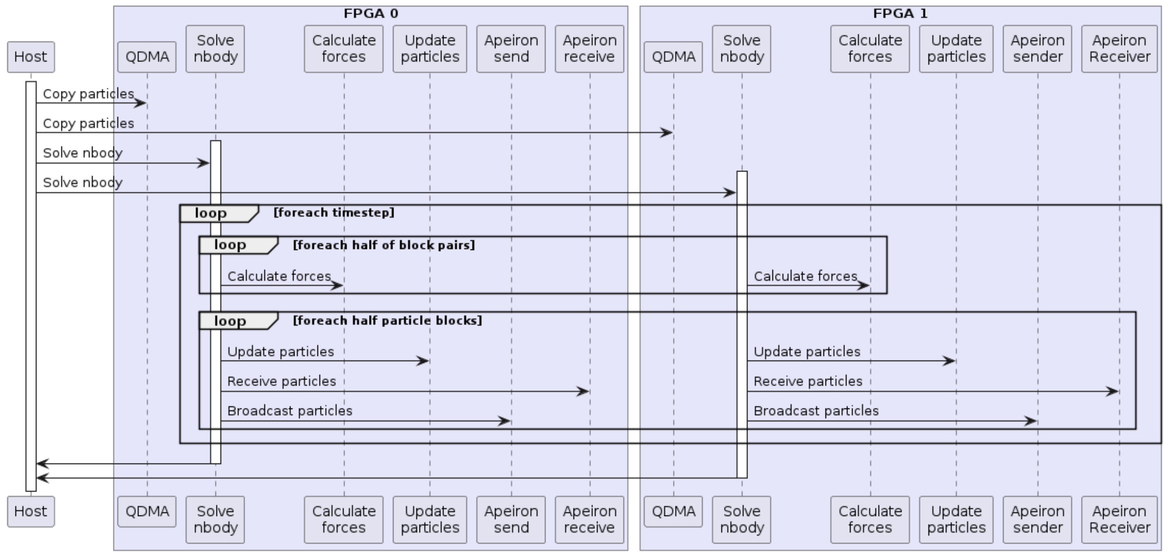


Figure 108: [NBody] Ompss + apeiron sequence diagram

This figure shows a sequence diagram showing a high-level view of the interaction of application accelerators and apeiron modules. This diagram shows the case of two FPGA devices connected to the host via PCIe using the QDMA module and between devices via QSFP using apeiron modules.

First, data is copied to all devices, then a single *solve nbody* task is submitted to each device.
Then, for each timestep, forces between each particle pair are computed. Each of the two nodes compute forces for half the particles. Then, velocity and position of the particles assigned to each node are updated. Finally, the updated particles are sent to all other nodes.
Data send and receive operations are usually done in parallel and data movements can be overlapped with task execution. Also, multiple *calculate forces* tasks are run in parallel in multiple accelerators, although not shown in the diagram for clarity.
Comparing with flow described in Figure 105 for the host directly managing multiple devices it is obvious that the number of data movements are greatly reduced when data can be transferred between devices without host intervention. Also, the number of communication operations related to task submission and synchronization involving individual task execution for each particle block is greatly reduced.
This approach allows performance to scale when using multiple devices, as shown in Figure 109.

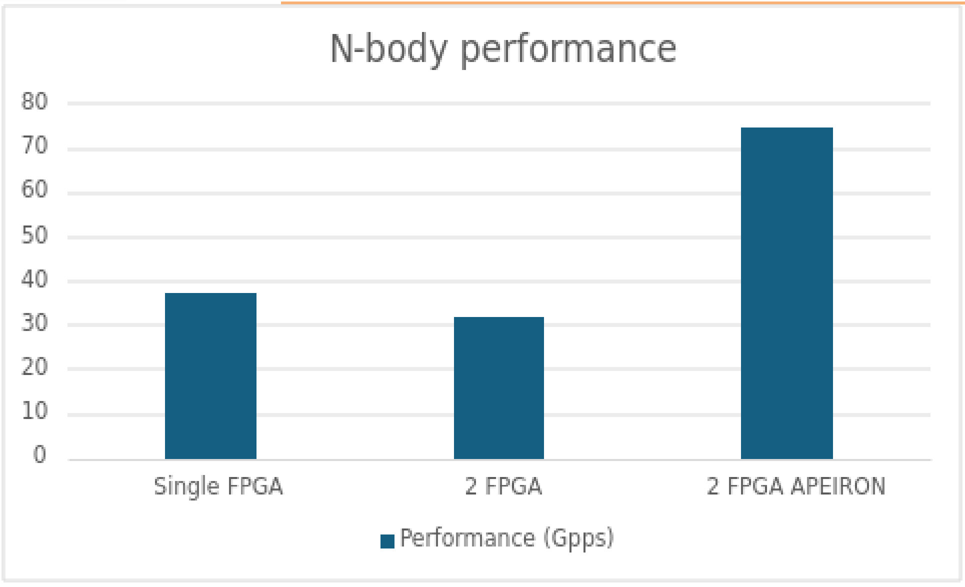


Figure 109: [NBody] Performance for different n-body FPGA implementations

The chart clearly shows that performance using apeiron for communications between both FPGAs performs much better than the previous host-centric approach. In fact, speedup achieved using this approach provides a speed up of 1.99x when compared to the single FPGA implementation. This is very close to the ideal speedup of 2x the results from doubling the available resources.

Also, power efficiency is maintained when moving from one to two FPGA devices. This is thanks to the low overhead caused by communications between FPGAs and between the host and FPGA devices. Energy efficiency results for different implementations are shown in Figure 110.

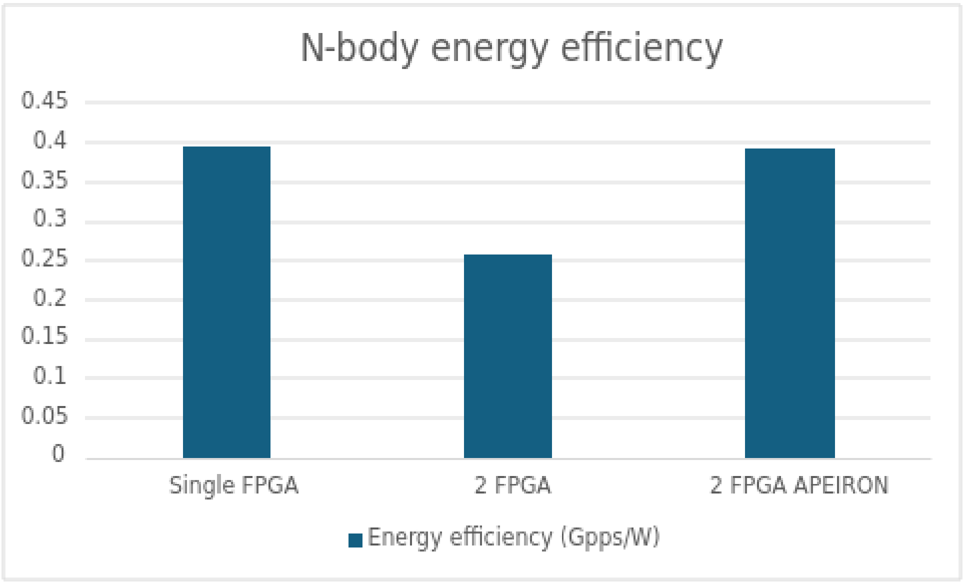


Figure 110: [NBody] Energy efficiency of n-body implementations

2.12.4 OmpSs@FPGA multinode applications

We evaluated the distributed n-body implementation across a larger number of devices.
The design follows the same approach as the implementation using apeiron described in section 3.12.3. Scalability tests are run in the MEEP cluster [14]. This cluster contains 96 Alveo U55c accelerator cards connected using 100G Ethernet installed across 12 different nodes. These cards use the same FPGA part as the Alveo U280, but without DDR. Also, the same application accelerators are used and therefore, per-FPGA performance should be the same as the IDV-E. In the MEEP system however, an external 100G ethernet switch is used instead of using the apeiron switch to route data packets to their destination.
Nevertheless, setup should provide a good approximation of the performance achievable by a cluster of IDV-E nodes and allows us to evaluate the distributed approach across multiple nodes with multiple FPGA devices each.

Tests have been carried using up to 64 FPGAs across 8 different nodes. Scalability results are shown in Figure 111.

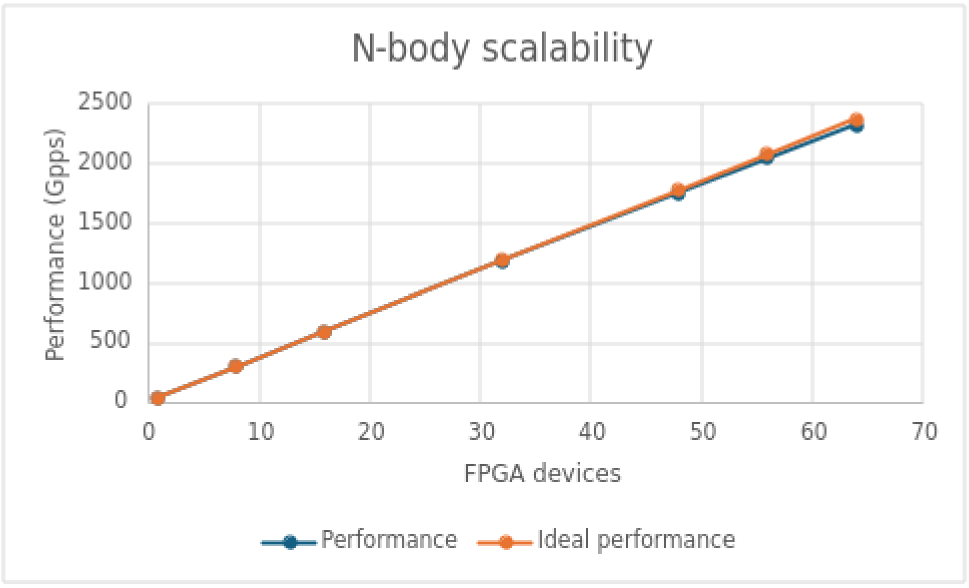


Figure 111: [NBody] Measured performance and ideal across different number of devices

This figure shows scalability results for the n-body application. For each number of nodes, it shows the actual, measured performance and the ideal performance. Ideal performance is the result of multiplying the performance of one FPGA by the number of devices.
Data presented in Figure 111 shows that measured performance very closely matches the expected performance. In the case of 64 FPGAs performance is 98% of the ideal performance. This demonstrates that the proposed distributed solution could still be efficient in a larger system comprised of several IDV-E nodes.

We compared scalability with a classical CPU-based system. The chosen system is MareNostrum 4, which as 3456 nodes with two Intel Xeon Platinum 8160 CPUs,
94GB of DDR4 RAM each, and 14nm Intel technology. It also features a 100Gb Intel Omni-Path Full-Fat Tree network. In our test, we measure performance for up to 56 nodes.
Results are shown in Figure 112.

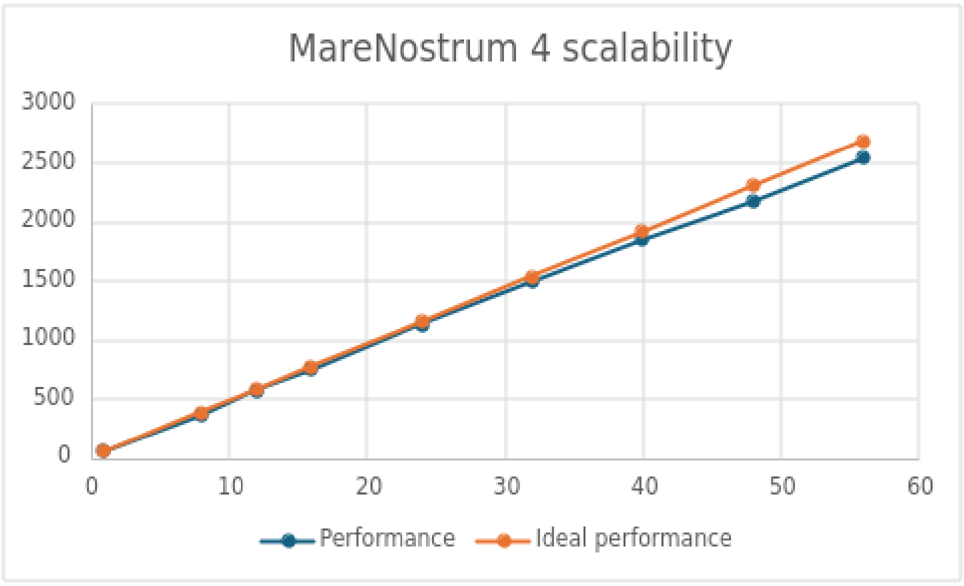


Figure 112: [NBody] MareNostrum 4 n-body scalability

This figure shows performance across different number of nodes. The chart shows two lines, one with the actual measured performance and another line with the expected ideal performance. Ideal performance is the product of the number of nodes by the performance on single node. While scalability is good, reaching 94.6% of the expected ideal performance, it's slightly lower than the 98% of the FPGA system.

Table 44 shows a summary of performance and power efficiency for IDV-E, MEEP multi-node FPGA system and MareNostrum4. The table reports the main KPIs (performance and power efficiency) obtained by the multinode system compared against the same numbers obtained in the MareNostrum4 supercomputer to show that the experimental system developed is able to compete with a current supercomputer in terms of performance while delivering more than 2x times better power efficiency even if the FPGAs technology is 16nm against the next generation 14nm MareNostrum4 processors.

Cluster (size)	Performance (Gpairs/s)	Power (W)	Power efficiency (Gpair/W)
IDV-E (1)	37.43	94.84	0.394
IDV-E (2)	74.69	189.64	0.393
MN4 (32)	1488.08	9459.0	0.157
MN4 (56)	2536.60	15504.2	0.163
MEEP(32)	1175.26	3227.9	0.364
MEEP(64)	2322.98	6472.7	0.359

Table 44: [NBody] Summary of performance, power consumption and power efficiency across different clusters

3 Summary and Concluding Remarks

In this deliverable we presented application benchmarks and results with respect to the KPIs defined in the D6.1 deliverable and as they were advanced throughout D6.2 up to the state that they reached on the final platforms. Next subsections detail the relevant results and for each application.

3.1 Smart Cities - CINI-UNIPI

The key lessons learned thanks to the work at application levels are:

1) Edge server applications for surveillance of smart cities can be set-up but to sustain in real-time combination of multiple complex algorithms like YoloV5+ Deep Sort a compute blade with a powerful GPU (Tesla A100) and processor (Xeon) should be adopted.
2) RISC-V computational capabilities, particularly in scalar version, and considering commercially available solutions are still far in performance from other platforms like those based on Intel and/or ARM.
3) For ARM the SVE version in Fujitsu performed worse than the ARM Neoverse N1 in Ampera Altra Max (the one in IDV-E) that is not using SVE; this can be justified by the fact that the algorithms were not optimized for a scalar vector extension version of the processor.
4) The availability of FPGA Xilinx Alveo in IDV-E can be exploited by porting the Yolo calculation on it while keeping the DeepSort on the ARM processor and hence a gain in speed can be achieved roughly by a factor of 2. However, the performance are still below those that can be achieved using Intel Xeon plus a GPU Like Tesla A100.
5) Posits alone, implemented via SW library emulated by the processor, but without a hardware support, cannot bring an advantage vs. architectures like Tesla A100 already supporting natively mixed-precision (FP64, FP32, BF16, INT8, INT4)

These results are important output for the European community working on defining an EU-based alternative to the monopolio of Intel/AMD plus NVIDIA GPU solution.
In future projects and benchmarking activity of CINI-UNIPI will be extended to consider also RISC-V architectures with Vectorized Instruction Set and compare them to ARM-based (Neoverse N1, A64FX) and Intel-based GPP architectures.

3.2 MathLib - CNR

CNR developed some multi-GPU kernels for efficient use of heterogeneous architectures embedding last generation of Nvidia GPUs, as in the IDV-A platform. Activities were focused on the design of new algorithms and implementation patterns which can exploit at the best the combination of distributed-memory/shared-memory programming models, leveraging multiple

GPU accelerators. Benefits of the library have been demonstrated both on IDVA and the Italian Leonardo supercomputer, accessed by an Early Access Grant obtained by the CNR team. The lesson learned is that for effective use of the new architectures it is not sufficient to re-factor legacy libraries and codes, but it is needed to rethink basic algorithms. The library was used as benchmark for the IDV-A platform and some project toolchain for energy consumption measurements before and after the installation of the new 2-phase cooling system, showing that this system has small impact on the performance results of the main kernel although can reduce the average power and then its total energy consumption.

3.3 RTM - Fraunhofer

Both Floating Point formats Posit 16 bit with one exponent bit as well as Float 16 bit demonstrated the potential to be successfully used for RTM calculations based on the Marmousi reference data set. Different scenarios are available. Common is the storage of the wavefield in reduced precision. This can be extended by also calculating the kernel in reduced precision as well as stacking the image in reduced precision. If the Kahan summation is used for the stacking procedure all the numerical operations can be executed very well in reduced precision. The deviation towards the reference solution can be improved by calculating the kernel in Float32 or stacking the image in Float32. This presumes that the core provides the capability to perform Float32 as well as reduced precision operations.

3.4 HEP - INFN

The comparative analysis of the different available architectures yields valuable insights on the KPIs (throughput and energy efficiency) for the applications of interest. For the CPU only version of both the applications we observed a better scaling in energy efficiency running on the Intel(R) Xeon(R) Platinum 8470 based platform (IDV-A) with respect to the Dibona AMD-based cluster. Obviously, we have to take into account that the IDV-A platform shows a baseline power consumption of around 400 W due to a 200 W consumed by the two CPUs when idle plus 200 W accountable to the thermal control daemon. So in absolute terms, in the current state of the IDV-A, the Dibona shows better results regarding energy efficiency. We expect that these results will be reverted after some optimization activities on the thermal control daemon.
On the other hand, the ARM-based IDV-E node proved more effective on both throughput and energy efficiency compared to the Dibona and IDV-A nodes. The IDV-E architecture reaches the same throughput of the more performant of the two X86_64 ones (IDV-A) while surpassing by roughly a 50% the more energy efficient of them (Dibona) in the energy efficiency KPI.

When considering the performance of the IDV-A platform in comparison with the Dibona one in the CPU+GPU configuration of the two HEP applications, the improvement in Pixeltrack throughput KPI is noticeable and expected in the IDV-A node . The slightly worse performance of CLUE on IDV-A for the throughput KPI compared to that measured on Dibona node needs further investigation, and possibly specific code tuning for the H100 GPU. Regarding the energy efficiency KPI, there is a slight advantage in using the Dibona node; again this hints for further investigations and possibly code tuning for the H100 architecture.

3.5 NEST-GPU - INFN

The comparison of the CPU and GPU versions of the *hpc_benchmark* test is smashingly in favor of the latter; in order to have a more complete reference against the D6.2 baseline this same multi-node benchmark is to be performed on the Dibona ARM cluster – the IDV-E used here is a single node so that inter-node communications were not gauged – but we do not expect a reversal. The same test is still to be performed on the state-of-the-art Intel CPUs of the IDV-A; on those, as in D6.2 we expect somewhat shorter runtimes at the cost of significantly increased energy-to-solution compared with those obtained on the ARM CPUs of the IDV-E mentioned in section 2.5 above but nothing close to the more than 17 times faster and 10 times smaller energy-to-solution a single IDV-A GPU can muster.

Even if the no-communications benchmark is faster than the multi-GPU ones the weak scaling that can be assessed with up to 4 GPUs of a single node IDV-A is quite good; for the accessible problem sizes that can be accommodated in the IDV-A in its current deployment the communications do not seem to be pose a significant hindrance yet – it remains to be seen for larger number of GPUs, possibly on more interconnected nodes – and there seems to be some performance to be squeezed out of optimizing the inter-GPU communications, as mentioned in section 2.4.

3.6 RAIDER - INFN

The further co-development of the application, of the APEIRON scalable HLS framework, and of the INFN Inter/Intra-FPGA Communication IP lead to significant improvements wrt the baseline reported in Deliverable D6.2.
In particular, in both testbeds used, we showed the scalability of the application in terms of throughput KPI, leveraging on the capabilities of the APEIRON framework in terms of multi-FPGA deployment and execution.
As can be extrapolated from the energy efficiency KPI values reported for the two testbeds, for this class of applications a good scaling in term of energy efficiency can be reached only if the added nodes have enough processing capabilities that allow to increase the integrated processing throughput at a higher rate of the additional energy consumption they introduce in the system. This is the case of the IDV-E testbed (1 IDV-E node with 2 Alveo U280 FPGA boards), while in the INFN APE-Lab testbed we found the opposite situation, where the increased processing throughput capabilities when scaling the number of nodes (1 Intel node with 1 Alveo U200 FPGA board) were not enough to compensate for the increased energy consumption in the system. This is due to the higher density of FPGAs per node in the IDV-E system (2 per node vs 1 per node), to the higher energy efficiency of the Ampere Altra processors compared to the Intel Xeon Silver counterparts in the APE Lab small cluster, and to the higher energy efficiency of the U280 FPGAs compared to the U200.

Regarding RAIDER integration with OmpSs, mixing task-based and streaming models can lead to increased performance with good programmability. However, tools and models need to be developed to exploit the advantages of both models. We also plan to further evaluate the integration of stream kernels into the task-based model. We plan to port the application to the

MEEP machine with multiple FPGAs (64) and take advantage of the FPGA-to-FPGA direct link communication with the APEIRON communication framework (to reach a set of 32 FPGA pairs) if the kernel requires more than one FPGA to be deployed with enough performance as the preliminary tests suggest.

3.7 TNM - INFN

Regarding the Quantum matcha TEA (gate-based quantum circuit emulator for digitized quantum circuit, the relevant KPIs were reported in deliverable D6.2 as a baseline for the IDV-A node performance. Unfortunately, we were not been able to run our application on the IDV-A yet given the short timeline of its availability, we plan to do it and report results in the next weeks.
For Quantum green TEA, i.e. solver for the Schrödinger equation or Lindblad equation, this is a new application developed after Deliverable D6.2. We reported a strong scaling study with multi-threading and MPI on single node towards finding the optimal split in hybrid multi-threading/MPI. The study was conducted using a node of the Justus2 cluster (bwHPC), with 2xIntel Xeon 6252 Gold with 2x24 cores, showing a ~5 speedup wrt the single core configuration. We pointed out the bandwidth towards the main memory as the main limiting factor to scalability, and we will repeat the same study on the IDV-A node in the next week and compare results. Furthermore, algorithmic improvements and comparison between performance on CPU and GPU on a CINECA Leonardo Booster node have been reported; the latter will be repeated on the IDV-A node in the next weeks.
The outcomes and knowledge gained during the TEXTAROSSA project for the TNM will be applied via the collaboration of the Quantum TEA team with HPC or via its open-source code. In 2023, we have been working together with CINECA and Quantum Matcha TEA was installed on *Leonardo*. Moreover, the software is open-source and will be made public step-by-step in the future such that features as the data type switching are available to the public. Moreover, the code is used for European projects as Pasquans2 or EuRyQa, Marie-Curie fellowships like the UniPhD program at the University of Padova, or the QRydDemo project funded by the German Ministry of Education and Research. The applications within these projects have and will profit from the intellectual property gained during TEXTAROSSA.

3.8 MathLib - INRIA

Our work focuses on task scheduling in heterogeneous architectures. Our novel scheduler can accommodate any combination of hardware types, including CPU/GPU and CPU/FPGA configurations, for instance. The primary advantage of our scheduler is its flexibility to be easily enhanced to improve the energy efficiency of task-based applications. We are actively developing this aspect, and our short-term objective is to demonstrate its benefits. Accordingly, our next milestone involves developing an energy-centric strategy and evaluating its performance on the Chameleon and ScalFMM platforms.

3.9 UrbanAir - PSNC

The work carried out focused on exploiting heterogeneous resources, namely GPU accelerators, by UrbanAir-gcrk. Comparing to execution on CPUs, the time-to-solution was decreased 9x (multinode enviornment), while energy efficiency was increased by 2x (single node). With further work, for the same accelerators up to 10% increase in iterations per second was achieved, and up to 22% increase in energy efficiency. It clearly demonstrates that with proper implementation GPU accelerators are bringing possibility to run more efficiently in terms of execution time and energy efficiency compared to execution on CPUs. While there is memory constraint on GPU limiting the maximum size of a problem to be solved, there is no such constraint in multiple node environment, where each accelerator can compute its portion of data, so that a significantly larger problem can be solved. The eventual gains in speedup and energy efficiency depend on the problem and algorithms used. For problems similar to UrbanAir-gcrk, where there are grid points in 3D space and the value of each is calculated at each iteration/timestep by the same scheme (e.g. exchanging data with its neighbors), the adaption to heterogeneous may be simpler and benefits easier to achieve. The reason for this is that in such cases the problem can be divided equally between accelerators, where each accelerator is computing their data independently, and just exchange the data with when neccesary at each timestep. It is important to remember that only fully utilized GPUs are bringing the maximum gain in computational performance (and thus time-to-solution).

3.10 FIPLib - ENEA

The aim of the study performed over this image processing application was to design a library, this is FIPLib, written in C++ and useful for a stream-based hardware design to be implemented on an FPGA board via the Vitis HLS tool. It has been proved that, taking into account the limited computing resources on the accelerator card, it is possible to synthetize a bitstream to be flashed on the FPGA capable of overperforming (in terms of throughput and energy consumption, as reported in the previous section) its CPU model counterpart.
In addition to this, exploiting the capabilities of the APEIRON framework built over the INFN Communication IP, it has been proved that it is possible to implement even larger (in terms of resources) image processing application thanks to the possibility of multi-FPGA deployment. The results presented in the previous section shows, in fact, how it is possible to implement FIPLib processing kernels over multiple boards maintaining the stream-based execution via the usage of the APEIRON HAPECOM Communication API. However, even if throughput performances seems to scale well with the number of boards in the setup, the energy consumption remains a parameter to take into account since the improvement in terms of processed image per second could result into a loss in terms of processed images per Joule with respect to the single FPGA implementation.

3.11 NBody - BSC

Evaluation of OmpSs@FPGA in diferent scenarios has highlighted that in task-based programming models, task management overhead can be critical for performance. It quickly can become a bottleneck, especially when using multiple accelerator devices as shown in section 3.12.2. Also, efficiently moving data between accelerator devices is critical when managing multiple devices, as shown in section 3.12.3.

Furthermore, proper cooling, which is often overlooked, can cause a non-negligible impact on energy efficiency and even on system usability if it is not properly addressed, as shown in section 2.12.1.
The work in the multinode capabilities of OmpSs@FPGA as shown in section 3.12.4, is expected to continue. Next steps include the inclusion of a Virtual Memory system in the OmpSs@FPGA framework to provide better scalability. Furthermore, we expect to leverage the Implicit Message Passing programming model, already under review, in a RISC-V manycore system with the help of the Fast Task Scheduler developed in Task 2.5. We plan to continue using NBody as a part of the test benchmarks used to evaluate these new developments along with other different applications (as maybe RAIDER, HPCG, etc.).

4 References

[1] Zenker, Erik & Worpitz, Benjamin & Widera, Rene & Huebl, Axel & Juckeland, Guido & Knüpfer, Andreas & Nagel, Wolfgang & Bussmann, Michael. (2016). Alpaka - An Abstraction Library for Parallel Kernel Acceleration. 10.1109/IPDPSW.2016.50.

[2] Emanuele Ruffaldi, Federico Rossi, Marco Cococcioni, "*cppPosit: a C++ template-based library implementing Posit arithmetic ready for machine learning applications*", https://www.techrxiv.org/doi/full/10.36227/techrxiv.24139605.v1

[3] 2021-05-06-NVIDIA Ampere Architecture In Depth NVIDIA Developer Blog.pdf (yyrcd.com)

[4] Heroux MA, Dongarra JJ. Toward a New Metric for Ranking High-Performance Computing Systems. Sandia National Lab. Tech. Rep., SAND2013-4744, 2013.

[5] Bernaschi M, Celestini A, D'Ambra P, Vella F. Multi-GPU aggregation-based AMG preconditioner for iterative linear solvers, 2023. IEEE Transactions on Parallel and Distributed Systems, vol. 34, 8, 2023. https://doi.org/10.1109/TPDS.2023.3287238

[6] NVIDIA, Algebraic multigrid solver (AmgX) library, rel.2.1, 2020. https://github.com/NVIDIA/AMGX.

[7] Naumov M, Arsaev M, Castonguay P, Cohen J, Demouth J, Eaton J, Layton S, Markovskiy N, Reguly I, Sakharnykh N, Sellappan V, Strzodka R. AmgX: a library for GPU accelerated algebraic multigrid and preconditioned iterative methods, SIAM Journal on Scientific Computing, 2015, 37, S602–S626.

[8] A Modular Workflow for Performance Benchmarking of Neuronal Network Simulations – Neuroinform., 11 May 2022 Volume 16 – **https://doi.org/10.3389/fninf.2022.837549**

[9] Extremely Scalable Spiking Neuronal Network Simulation Code: From Laptops to Exascale Computers – Front. Neuroinform., 16 February 2018 Volume 12 – **https://doi.org/10.3389/fninf.2018.00002**

[10] Runtime Construction of Large-Scale Spiking Neuronal Network Models on GPU Devices – Appl. Sci. 2023, 13(17), 9598 – **https://doi.org/10.3390/app13179598**

[11] Ballarin, M., Silvi, P., Montangero, S., & Jaschke, D. (2024). Optimal sampling of tensor networks targeting wave function's fast decaying tails. arXiv preprint arXiv:2401.10330.

[12] Xiz 2005] Xizhou Feng, Rong Ge, K.W. Cameron: Power and Energy Profiling of Scientific Applications on Distributed Systems. Proceedings. 19th IEEE International Parallel and Distributed Processing Symposium, 2005.

[13] J. M. de Haro *et al*., "OmpSs@FPGA Framework for High Performance FPGA Computing," in *IEEE Transactions on Computers*, vol. 70, no. 12, pp. 2029-2042, 1 Dec. 2021, doi: 10.1109/TC.2021.3086106.

[14] K. DeVogeleer, G. Memmi, P. Jouvelot and F. Coelho, "Modeling the temperature bias of power consumption for nanometer-scale CPUs in application processors," *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, Agios Konstantinos, Greece, 2014, pp. 172-180, doi: 10.1109/SAMOS.2014.6893209.

[15] MEEP: Marenostrum Exascaleme Entry Project, D6.4 -Full Emulation prototype release.