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

WP6 Applications and Use cases

D6.2 Initial Application Benchmarks and Results

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: D6.2 Initial applications benchmarks and results

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	D6.2
WP No:	WP6
WP Leader:	PSNC
Due date:	M24 (March 31, 2023)
Delivery date:	31/03/2023

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:	D6.2					
Deliverable title:	Initial applications benchmarks and results					
Due date:	M24					
Actual submission date:	03/04/2023					
Editor:	Michał Kulczewski					
Authors:	Massimo Bernaschi, Berenger Bramas, Laura Cappelli, Alessandro Celestini, Pasqua D'Ambra, Francesco Giacomini, Daniel Jaschke, Martin Kuehn, Michał Kulczewski, Alessandro Lonardo, Alice Pagano, Cristian Rossi, Sergio Saponara, Francesco Simula					
Work package:	WP6					
Dissemination Level:	Public					
No. pages:	81					
Authorized (date):	31/03/2023					
Responsible person:	Michał Kulczewski					
Status:	Plan	Draft	Working	Final	[✓ Submitted]	Approved

Revision history:

Version	Date	Author	Comment
0.1	2023-01-18	Michał Kulczewski	Draft structure
0.2	2023-03-03	Sergio Saponara	Filled CINI part for the smart cities use case
0.3	2023-03-10	Pasqua D'Ambra	CNR-MathLib section has been included
0.4	2023-03-16	Michał Kulczewski	UrbanAir section has been included
0.5	2023-03-22	Martin Kuehn, Laura Cappelli, Alessandro Lonardo, Francesco Simula, Daniel Jaschke, Berenger Bramas, Cristian Rossi	All sections have been included

0.6	2023-03-23	Michał Kulczewski	Ready for internal review
0.7	2023-03-29	All	Addressing comments
0.8	2023-03-30	Michał Kulczewski	Ready for submission

Quality Control:

Checking process	Who	Date
Checked by internal reviewer 1	Francesca Lo Cicero	2023-03-27
Checked by internal reviewer 2	Carlos Alvarez	2023-03-28
Checked by Task Leader	Pasqua D’Ambra	2023-03-30
Checked by WP Leader	Michał Kulczewski	2023-03-30
Checked by Project Coordinator	Massimo Celino	2023-03-31

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 http://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 provides initial benchmarks and results of the applications. In particular, the document discusses outcomes with respect to KPIs defined for each use case in the previous D6.1 deliverable. 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 (Dibona node with GPU accelerators from Atos) and IDV-E (node with FGPA accelerators from E4). The applications represent wide range of application types (HPC, AI/HDPA), scientific domains, approaches to parallelization (OpenMP, CUDA, MPI, …). 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 application of smart cities is useful in case of disaster (e.g. earthquakes) or terrorist attacks or war scenarios. It contains an AI+video algorithm to detect people in a scene and then detecting and tracking people that are laying down. Input images acquired by video-camera are passed to a YOLOv5 (You-look-only-once) detection system, and its outputs are passed then to a filtering step detecting people that are laying down. A DeepSORT algorithm that implements tracking and counting tasks is finally applied. DeepSORT is an extension of the SORT (Simple Online Realtime Tracking) algorithm. This algorithm is tested on several HPC platforms using ARM and INTEL GPPs, with and without accelerators, some of them supporting mixed-precision.

CNR designed and implemented a large part of the proposed Mathlib kernels, which includes widely used building-blocks for physics-driven simulation models in traditional HPC applications. Some new algorithms specifically thought for heterogeneous computing nodes hosting Nvidia GPUs have been proposed and efficient parallel design patterns have been applied to obtain high performance at the node level and large scalability. Measured KPIs on some Textarossa platforms and some Top 500 supercomputers demonstrate the validity of our approaches. A prototype of the CNR Mathlib is already available in a public repository and it can be considered one of the key innovations of the Textarossa software toolchain for performance/energy efficient computations.

Fraunhofer extended a basic implementation of a 3D isotropic RTM Kernel using 5 different floating point formats of reduced precision to compress the domain and the model as well. The focus is to save memory bandwidth to increase runtime performance while keeping up the image quality. Images calculated with 6 different approaches have been compared quantitatively and qualitatively. Some formats like Bfloat16 proved not suitable for seismic applications while other formats like Posit16 or ZFP provided an acceptable image quality. Furthermore the numerical stability of the suitable implementations have been tested versus

the applied timestep including two difficulty levels in the velocity model. Most implementations revealed less than 5% drop in the stable timestep size.

INFN provides one HPDA (RAIDER) and three HPC (TNM, NEST-GPU, HEP) applications as benchmarks to drive the co-design and characterization activities of project IDVs.
We tailored the RAIDER (Real-time AI-based Data analytics on hEteRogeneous distributed systems) application for the use case of the CERN NA62 High Energy Physics experiment in order to demonstrate the effectiveness (in terms of processing throughput and energy efficiency KPIs) of our scalable streaming based framework (APEIRON) in addressing three project objectives: i) *Seamless integration of reconfigurable accelerators*: the APEIRON framework allows to integrate a number of communicating HLS kernels (coded in C++) both on a single FPGA or on a set of several interconnected FPGAs, ii) *Energy efficiency*: first results on energy efficiency show a O(10) improvement when compared to CPU or CPU+GPU implementations, and iii) *Sustained application performance*: first results on processing throughput show a x3 and x2 improvement respect to CPU and CPU+GPU respectively with a limited exploitation of the scalability features offered by the APEIRON framework and using just 20% of a single Alveo U200 FPGA resources. The key enabling technology behind the APEIRON framework is the INFN intra/inter-FPGA communication IP, developed according to the project objective *Development of new IPs.*
Tensor Network Methods (TNM) application for the simulation of quantum systems is used to emulate the behaviour of quantum computers.
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 HPC platforms.
Finally, regarding High-Energy Physics (HEP) codes on heterogeneous architectures we have selected two representative applications: Pixeltrack, a track reconstruction algorithm for the CERN CMS experiment, and CLUE, a cluster algorithm for high-granularity calorimeters. The topics related to the three HPC applications are important in current and future HPC scenarios, and the applications represent relevant benchmarks to characterize and possibly shape the architecture of TEXTAROSSA IDVs.

INRIA created a new scheduler called Multreeprio within StarPU. This scheduler is highly modular, as each task can have several priorities, one for each type of processing unit, allowing for the creation of compact heuristics to favor locality, makespan, or energy efficiency. We evaluated this scheduler on two applications, ScalFMM and Chameleon, using different types of heterogeneous computing nodes. We measured two key performance indicators (KPIs): Flops/s and Flops/Watt.

PSNC provides GCRK kernel, one of the main routin of EULAG model. It was tailored towards the use in UrbanAir application, dealing with air quality forecasting. The work focused on addressing two objectives: energy efficiency and application performance by measuring proposed KPIs: iterations/s and iterations/Watt. In the former case, we are able to achieve 3.5-9x speedup comparing CPUs to GPUs, but it depends on the problem size and amount of hardware resources used. In the latter case, we can achieve at least 2x more energy efficient run, although the number depends on the problem size and amount of hardware resources used.

1 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 development paradigms can improve computation and energy efficiency of applications coming different domains.

In the Textarossa we focus on 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 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 advancements in applications development, and on reporting initial benchmarks and results to steer further development. 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, using scheduler of streaming framework;
- 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 (Dibona) and IDV-E platforms for initial benchmark results;
- Opening of new usage domains, by developing application in many different domains, e.g. climate, oil&gas, high energy physics.

This document is organized as follows. In Section 2, methodology for benchmarking applications from computational and energy efficiency perspective is discussed. It also details hardware architectures being used for testing. In Section 3, each use case is described in details, providing its status, development advancements, and initial benchmarking results including KPIs. Section 4 provides a summary and discusses next steps.

2 Methodology

For the consistency of benchmarks results across different applications in the project, a common methodology is proposed to be obeyed by each of the use case. Each application provides results on the currently available testbeds, IDV-A (Dibona) equipped with GPUs, and IDV-E equipped with FGPA accelerators. Additionally, some applications performed additional benchmarks on external to the project hardware, which is detailed in Section 2.1 and within individual applications results. In Section 2.2, methodology to measure computational performance and energy efficiency is discussed.

2.1 Hardware

2.1.1 IDV-A

The IDV-A prototype defined in Textarossa is a GPU blade provided by Atos for two-phase liquid cooling adaptation. This prototype has been described in Deliverable D1.2 Chapter 3 (GPU platforms requirements). It will be available when this new liquid cooling is installed and validated on this blade. Unfortunately, the cooling design is still ongoing, and the prototype delivery targeted in M18 (Deliverable D5.1) has been delayed. Meantime access has been provided to a GPU node of previous generation in the Dibona cluster accessible to Atos partners in several funded projects.

This node is implemented in a CRRM blade in BullSequana XH2000 platform. The architecture of this node, described in Figure 1, is based on the following components:

- One bi-socket host with AMD EPYC 7402, codename "Rome" (8 DDR4 memory channel @3200 MT/s and 2 x16 PCIe Gen4 slots per socket).
- 4 Nvidia GPU Ampere A100, interconnected with NVlink
- Two 96-port PCIe Gen3 switches to provide direct access between CPU and GPU, CPU and NIC, GPU and NIC (GPU direct)
- Up to four NICs (Network Interface Controller) to connect to a HPC high speed interconnect (Infiniband HDR technology).

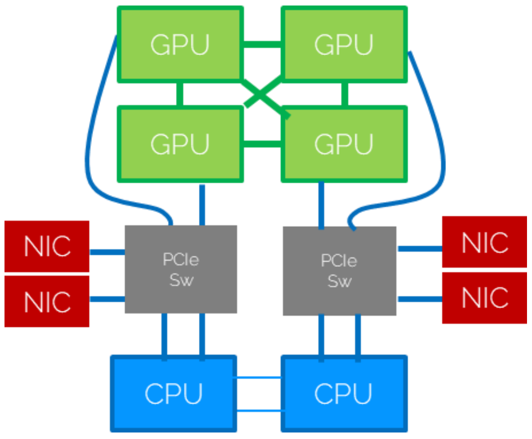


Figure 1 GPU blade architecture with PCIe switches

In the case of Dibona platform, only one blade is provided, then there is no connection to high-speed interconnect. The only access is the 1Gb/s link of the CPU host.

The architecture of final IDV-A is similar, except that the PCIe switch is embedded in the Infiniband NDR NIC (ConnectX 7 ou CX7), as described in Figure 2.

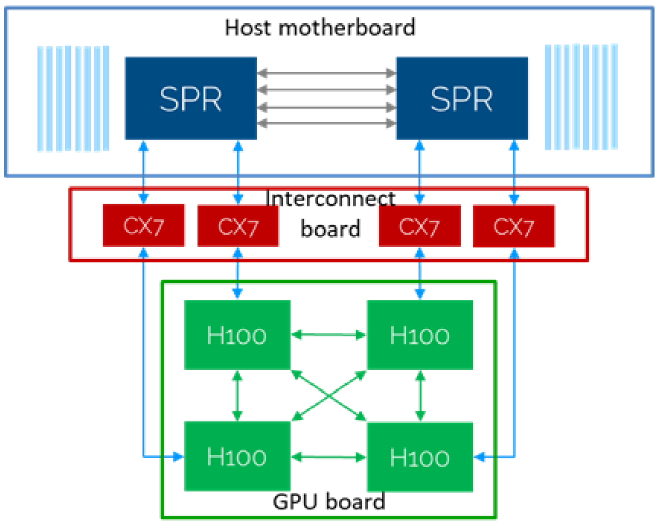


Figure 2 GPU blade architecture with embedded PCIe switches

This architecture change has no impact on the blade performance. But the performance will increase with these two main evolutions:

- Nvidia GPU is the next generation Hopper H100, interconnected with Nvlink.
- In the host node, the AMD Rome CPU is replaced with Intel Sapphire Rapids CPU, with 8 DDR5 memory channels @4800 MT/s and 2 x16 PCIe Gen5 slots per socket.

This blade will be hosted on the new BullSequana XH3000 platform. As this blade remains isolated, the node will also be accessed with the 1Gb/s Ethernet link of the host.

More details can be found in D1.4.

2.1.2 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 software stack. More details are described in D5.2 and D1.4.

2.1.3 Other architectures

For CNR-MathLib, Piz Daint system is used, operated by the Swiss National Supercomputing Center. That system is based on the Cray XC40/XC50 architecture with 5704 hybrid compute nodes (Intel Xeon E5-2690 v3 with Nvidia Tesla P100 accelerator) and 1813 multicore compute nodes (Intel Xeon E5-2695 v4), using the Cray Aries routing and communications ASIC with Dragonfly network topology. It is ranked 26th in the November 2022 Top 500 list.

For the Smart Cities applications by CINI in Section 3.1, CINI has considered multiple platforms:

- platform 1 --> **Quad core ARM Cortex-A72 on a Raspberry PI4 board**
- platform 2 --> **ARM Neoverse N1 (80 cores) on Ampera Altra blade**
- platform 3 --> **Fujitsu A64FX (ARMv8-A based) on an Apollo HPE cluster**
- platform 4 **-->** **NVIDIA Jetson AGX Orin**
- platform 5 --> **Intel i7-10750H with NVIDIA GeForce GTX 1650 Ti**
- platform 6 --> **Intel Xeon with NVIDIA Tesla T4**
- platform 7 --> **Intel Xeon with NVIDIA A100**

The platforms include processors based on ARM64 architecture (platform 2 and platform 3) with and without SVE extension.
The platforms include also Intel i7 and Xeon processors, and use also of different type of accelerators (T4, A100, Orin).

The platform with A100 supports mixed arithmetic like BF16/FP16/FP32/T32.
T different formats of the mixed-precision are presented in Figure 3.

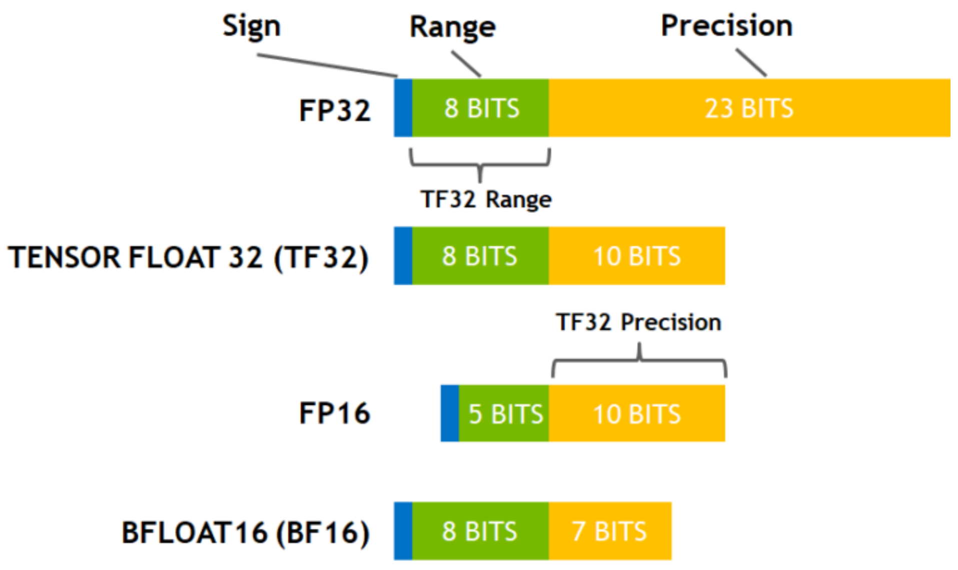


Figure 3 Different mixed-precision format supported by accelerator in platform 7 (with A100)

FP16/FP32 mixed-precision is also supported by GPU T4 that is one of the used platforms.
Missing an HW support to Posit in the available accelerators, the mixed-precision included BF16/FP16/FP32 but not Posit.
All accelerator architectures considered for the smart cities CINI use cases also support INT4/INT8.

Being the platform equipped with a FPGA, IDV-E is the reference platform of the RAIDER application by INFN in the context of the TEXTAROSSA project. Since the RAIDER application is developed using the INFN APEIRON framework, which is based on the Vitis flow, it relies on the XRT runtime libraries and the corresponding XOCL/ZOCL device driver. Xilinx does not support XRT on ARM platforms other than ZynQ, as is the case for IDV-E that integrates a Xilinx Alveo U280 on an ARM server. An activity from project partner BSC is finished to add the needed support, as reported in deliverable *D4.1-Proof-of-concept Textarossa IDV-E Test support*. However, the issues were solved too late to be accounted in this deliverable. Therefore results reported in this deliverable have been collected using a development platform available in the INFN Roma APE Lab, a dual socket server with two X86_64 processors (Intel(R) Xeon(R) Silver 4208 CPU @ 2.10GHz) equipped with a Xilinx Alveo U200 FPGA card. In section 3.6 we compare the KPIs obtained on the CPU+FPGA setup with a baseline given by the execution of the inference task on the Tensorflow model running both on CPU only and in combination with a GPU accelerator. These baseline KPIs have been collected using a workstation sporting a single socket Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz 8 core processor (with Hyper Threading support) and a NVIDIA GeForce RTX 2080 Ti GPU accelerator.

For the UrbanAir application, PSNC Altair supercomputer is used. It is a CPU and GPU cluster equipped with Intel processors: Xeon E5-2697 (2342), Xeon E5-2682 (110), Xeon Platinium 8268 (2640), Xeon Gold 5115 (6), Xeon 6242 (18), and Nvidia V100 cards. The Altair is ranked 158th in TOP500 list.

2.2 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 deliverable. 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).

3 Results

3.1 Smart cities – CINI-UNIPI

The application of smart cities contains a video-based algorithm to detect people in a scene and then detecting and tracking people that are laying down. This application is useful in case of disaster (e.g. earthquakes) or terrorist attacks or war scenarios. The input images acquired by video-camera operating in the visual domain are passed to a YOLOv5 (You-look-only-once) detection system, that is one of the most popular object detection algorithms. The outputs of YOLOv5 are passed then to a filtering step detecting people that are laying down. Then a DeepSORT algorithm that implements tracking and counting tasks is applied.
DeepSORT is an extension of the SORT (Simple Online Realtime Tracking) algorithm.
Algorithm inputs are:

- YOLOv5 model weights
- Re-Identification model weights
- Source path (path of the video file that should be processed)

Algorithm outputs:

- videos processed with detected and tracked people

The platforms tested are:

- platform 1 --> **Quad core ARM Cortex-A72 on a Raspberry PI4 board**
- platform 2 --> **ARM Neoverse N1 (80 cores on Ampera Altra blade)**
- platform 3 --> **Fujitsu A64FX (ARMv8-A based) on an Apollo HPE cluster**
- platform 4 --> **NVIDIA Jetson AGX Orin (12-core Arm Cortex-A78AE plus GPU NVIDIA Ampere)**
- platform 5 --> **Intel i7-10750H with NVIDIA GeForce GTX 1650 Ti**
- platform 6 --> **Intel Xeon with NVIDIA Tesla T4**
- platform 7 --> **Intel Xeon with NVIDIA A100**

Platforms 1, 2 and 3 are homogenous multi-core platforms using different types of ARM cores. Platforms 4, 5, 6 and 7 are heterogenous platforms with multi-core ARM or Intel GPP plus a GPU-based accelerator.

Table 1 reports the achieved results. In this tasble we use this legend to clarify if for the heterogenous platforms (4 to 7) we have used CPU-only or both CPU and the GPU accelerator:

Table 1 🐢 execution on CPU only
🚀 execution on CPU and GPU
NM --> Not Measured

To be noted that the results in terms of processing time for one frame are splitted among the YOLO inference phase, the mandown classifier and the DeepSORTphase. The frame per seconds is calculated as inverse of total time to process one frame. The temperature and power consumption of the chips (CPU and/or GPU) are also reported.
The results achieved demonstrate for video surveillance application the importance of accelerating the GPP with e.g., a GPU.

Indeed, the only platforms achieving a real-time are those with a GPP (e.g. Intel Xeon) plus GPU T4 or A100.
To be noted that platforms 2 and 3 are representative of the ARM HPC cores that will be available in the GPP of the European Processor Initiative. Comparing speed results of platform 2 and platform 3, is clear that the support of the Scalable Vector Extension in ARM (present in the Fujitsu A64FX of platform 3, while missing in the ARM Neoverse of Platform 2)   allows for a speed increase by a factor at least 2.

In terms of used arithmetics:

- platforms 1,2 and 3 adopt a classic fp32 floating-approach;
- platform 7 sustains fp32, fp16 and bfloat16;
- platform 4 sustains float32 and int8 for platform;
- platform 6 sustains ; fp32, fp16, int8, int4 for platform 6.

Platform	FPS	YOLO Inference Speed (ms)	Man Down Classifier Speed (ms)	Deep SORT Speed (ms)	CPU Temp (°C)	CPU Power Consumption(W)	GPU Temp (°C)	GPU Power Consumption(W)
1	0.1	7632	1.2	1032	82.0	NM	-	-
2	0.2	879	0.6	3794	51.6	NM	-	-
3	0.4	1221	1.1	1233	NM	NM	-	-
4 🐢	0.05	2084	0.5	18155	58.2	16.8	-	-
4 🚀	7.7	38.8	0.5	54.9	52.2	6.7	47.0	16.0
5 🐢	1.0	794	0.3	200	95	NM	-	-
5 🚀	6.5	82.9	0.3	37.9	NM	NM	74.7	38.2
6 🐢	1.8	336	0.3	197	NM	NM	-	-
6 🚀	11.7	33.9	0.3	28.3	NM	NM	48.9	57.1
7 🐢	2.0	328	0.3	191	NM	NM	-	-
7 🚀	16.3	16.2	0.3	25.3	NM	NM	30.8	44.4

Table 1 Results of target application (Yolo+DeepSort) on the 7 target platforms (with those with accelerator data with GPP only or GPP+accelerator are reported)

3.2 MathLib – CNR

In this section we discuss some preliminary results obtained by using the mathematical software library for heterogeneous architectures, featuring NVIDIA GPUs at node level. As already described in D6.1, the CNR team is developing computational kernels required in sparse matrix computations and iterative linear solvers, which are widely exploited in Scientific Computing and Data Analysis. Main focus is both on node-level efficiency and on scalability when multiple nodes are needed for computations because dimensions largely exceed the memory resources of a single computing node. The MathLib computational kernels developed and tested in the first phase of the project are the following:

1. Sparse matrix – vector multiplication (SpMV);
2. Sparse matrix – matrix multiplication (SpMM);
3. Maximum Weight Matching in undirected graphs (MWM);
4. Preconditioned Conjugate Gradient (PCG) method coupled with a matching-based Algebraic MultiGrid preconditioner (BCMG).

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). In the following we first discuss preliminary results obtained on the CRRM Blade for the Textarossa project, named Dibona and operated by ATOS. Then, in order to analyze the scalability potential of our main kernel, that is the sparse linear solver based on the AMG-PCG method (BCMG), we also show weak scalability results obtained on Piz Daint. As benchmark datasets 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 right-hand side equal to the unit vector. This is a standard benchmark test case for sparse matrix computation because it represents the computational kernel of many scientific and engineering applications, and indeed is also used in the HPCG benchmark [1]. 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. All the data are real and represented in double-precision format. 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.

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 MPI tasks. Blocks of contiguous rows are assigned to each task according to the order defined by the MPI rank. We observe that each MPI task is associated to one GPU accelerator which is in charge of all the computation phases. Details on the algorithms and parallel design patterns implemented for all kernels of the MathLib are discussed in [2]. Here we just mention that some approximations in some original (mono-GPU) numerical algorithms have been required, so that the combination of communication-avoiding techniques, fine-grained parallelism, and overlapping between computation and data communication could allow us to design a scalable version of the BCMG linear solver. 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.

3.2.1 Preliminary results on Dibona

In this section we discuss results of different kernels, in terms of both parallel performance and power consumption. For strong scalability analysis of the kernels, we always consider matrices with size 300^3=27M, whereas for weak scalability we consider different matrix dimension per each kernel, so that each GPU can be used at full load, as detailed in the following. Power consumption measures have been obtained for all the kernels when they run on 1 GPU, while in the case of multi-GPU executions, we obtained measures only for the whole BCMG solver in the weak scaling approach, when the GPUs are used at their full load. As also described in deliverable D6.1, our main KPIs are grouped in 3 main categories, as classified in Table 2.

KPI for computational efficiency	KPI for energy	KPI for accuracy
- execution time - (strong and weak) speedup - number of iterations/time per iteration (only for iterative linear solver) -accuracy	- iterations/Watt (only for iterative linear solver) - Dof (Degrees of Freedom or unknowns)/Watt	Yes (User's parameter dependent for iterative linear solver)

Table 2 KPIs for MathLib

3.2.1.1 Strong Scalability on Dibona

In the following, we discuss KPIs for computational efficiency, in particular we focus on execution time and strong speedup scalability, having as baseline the mono-GPU version of the kernel. Accuracy of the results of all, but the linear solver, kernels is up to the machine precision in double precision floating-point arithmetic. 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.

We first discuss performance of the SpMV kernel. It is a BLAS-2 operation involving a sparse matrix. BLAS-2 operations are characterized by a low intensity operation, indeed the ratio between number of flops and data access is constant for increasing matrix dimension. Therefore, their performance is limited much more by memory and communication bandwidth than the floating-point capabilities of the architecture. On the other hand, this feature is emphasized when the matrices are sparse, with a very small number of non-zero entries per row. In Figure 4 (top) we can observe that the execution time of the SpMV kernel decreases with increasing number of GPUs resulting in a speedup (bottom of Figure 4) of 1.44x on 2 GPUs and of 2.80x on 4 GPUs.

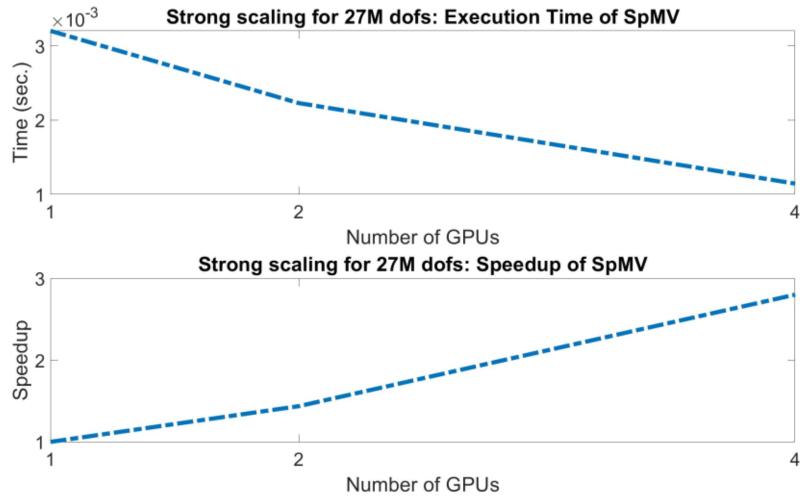


Figure 4 Strong Scalability SpMV kernel: Execution time (top) and Speedup (bottom)

In Figure 5 we show execution time (top) and speedup (bottom) of the SpMM kernel. Here the main issue is represented by the number of non-zero entries as well as the sparsity pattern of the resulting matrix product that are not predictable in advance. A so-called *symbolic* phase is in general applied, in which the number of nonzeros in the result matrix is computed, postponing the actual calculations of the values to a following *numeric* phase. For the single GPU version, we resorted to the *nsparse* [3] package, which revealed much more efficient of any combination of primitives provided by Nvidia's *cuSparse* library. Assumed that both operands are distributed in a consecutive row-block setting assigned to parallel tasks with consecutive MPI ranks, a straightforward solution to split the product computation among the GPUs is that each GPU computes the corresponding block of rows of the product matrix. In this setting, data communication among all the tasks is needed. The data communication volume and pattern depend on the nonzeros entries of the local rows, whose column indices correspond to rows, owned by other tasks, which have to be communicated to finalize the local computation on each GPU. We exchange all the data necessary to complete the product on each GPU before starting the computation, so that the product appears as if it were completely local from the viewpoint of the *nsparse* CUDA kernel. From Figure 5 we can see that execution time of SpMM, as expected, decreases for increasing number of GPUs and the speedup is 1.43x on 2 GPUs and 1.69 on 4 GPUs. The reduced speedup on 4 GPUs is due to the increased impact of data communication in this kernel with respect to the SpMV kernel.

In Figure 6 we show execution time (top) and speedup (bottom) of the MWM kernel, that is the computation of an approximate maximum weight matching in undirected weighted graphs. In this case we applied an embarrassingly parallel approach, where non-local edges are neglected, so that each task can compute an MWM approximation to the local sub-graph. Therefore, we can see that almost ideal speedup of about 2x and 4x, respectively on 2 and 4 GPUs, is obtained.

In Figure 7 and Figure 8 we show performance of the BCMG code which includes the setup of an AMG preconditioner and of the application of the corresponding preconditioned Conjugate Gradient method for solving linear systems arising from the Poisson problem, when a fixed number of 27M unknowns is considered. We can observe that in both cases the execution time is reduced for increasing number of GPUs and that a speedup of 1.39x and 1.67x, respectively on 2 GPUs and 4 GPUs, is obtained for the preconditioner setup, while a speedup of 1.30x and 1.87x, respectively on 2 GPUs and 4 GPUs, is obtained for the preconditioned Conjugate Gradient application.

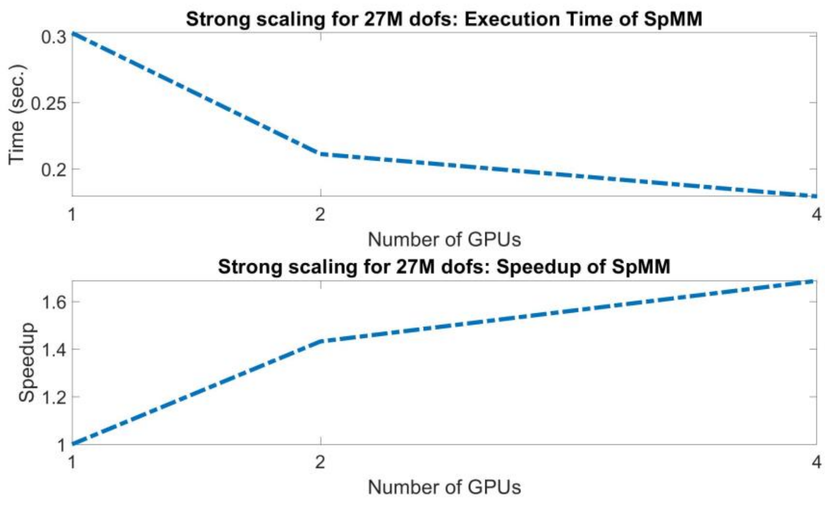


Figure 5 Strong Scalability. SpMM kernel: Execution time (top) and Speedup (bottom)

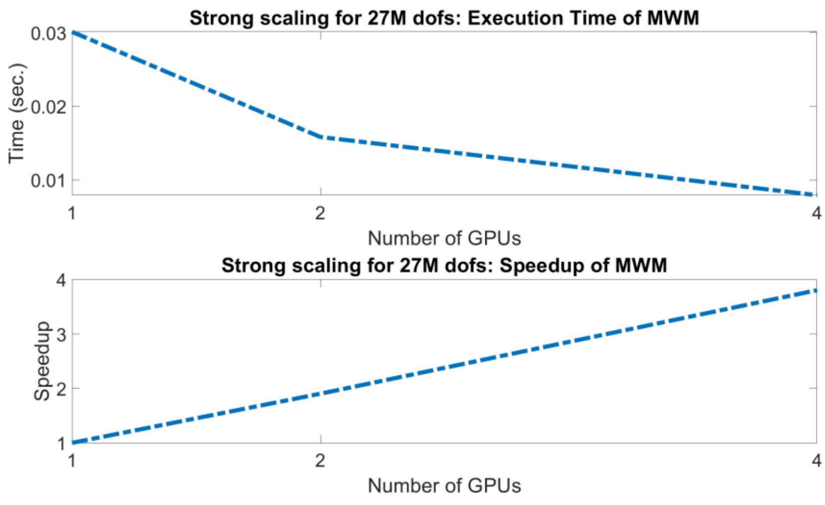


Figure 6 Strong Scalability. MWM kernel: Execution time (top) and Speedup (bottom)

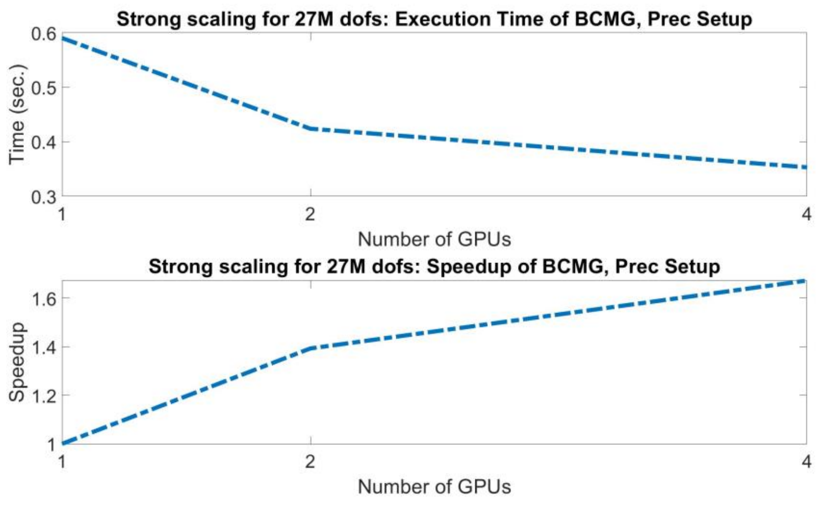


Figure 7 Strong Scalability. BCMG Preconditioner setup: Execution time (top) and Speedup (bottom)

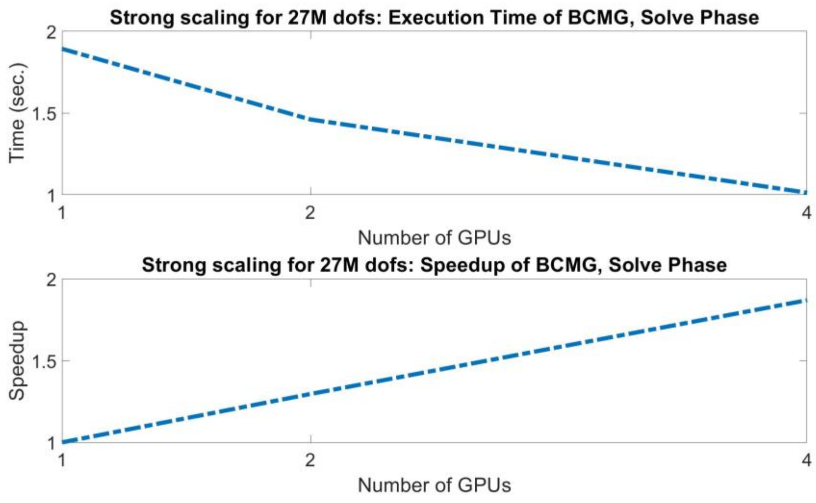


Figure 8 Strong Scalability. BCMG Solve: Execution time (top) and Speedup (bottom)

3.2.1.2 Weak Scalability on Dibona

As already mentioned, here we present some weak scalability results in using up to 4 GPUs available on the Dibona cluster for all the kernels of our MathLib. We used different problem sizes per each GPU for the different kernels, so that, per each kernel, the GPU are used at the full load. As in the strong scalability analysis, we discuss KPIs for computational efficiency, in particular we focus on execution time and weak (scaled) speedup, having as baseline the mono-GPU version of the kernel. In more details, we defined the scaled speedup as the ratio T_1(N)*np/T_np(np*N), where T_1(N) is the execution time for solving a problem with dimension N on 1 GPU and T_np(np*N) is the execution time for solving a problem with dimension np*N on np GPUs. Accuracy of the results of all but the linear solver, kernels is up to the machine precision in double precision floating-point arithmetic. In the case of the BCMG linear solver, 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.

The SpMV kernel obtains very good scaled speedup (Figure 9, bottom) of 1.76x and 3.32x, respectively on 2 and 4 GPUs. On the other hand, SpMM has a good scaled speedup (Figure 10, bottom) of 1.78x on 2 GPUs, while a scaled speedup of 2.23x is observed on 4 GPUs, showing that with this data sets, the data communication impact increases for increasing number of GPUs. MWM, as expected, shows an ideal speedup (Figure 11, bottom) of 2x and 4x, respectively on 2 and 4 GPUs, due to its embarrassingly parallel nature. As to the BCMG preconditioner setup we observe a scaled speedup (Figure 12, bottom) of 1.75x on 2 GPUs and of 3.28x on 4 GPUs. Finally, in the solve phase of BCMG, we observe a scaled speedup (Figure 13, bottom) of 1.50x on 2 GPUs for solving a sparse linear system with 78M of unknowns, and of 2.24x on 4 GPUs for solving a linear system with 156M of unknowns.

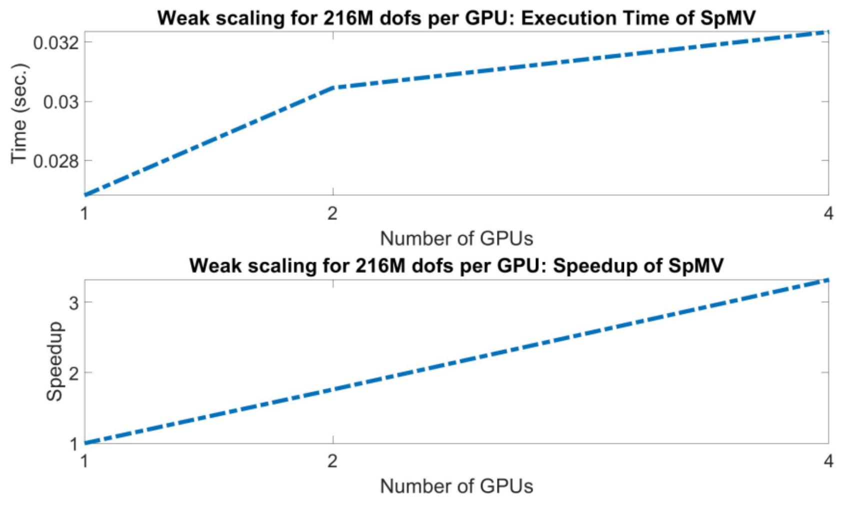


Figure 9 Weak Scalability. SpMV kernel: Execution time (top) and Speedup (bottom)

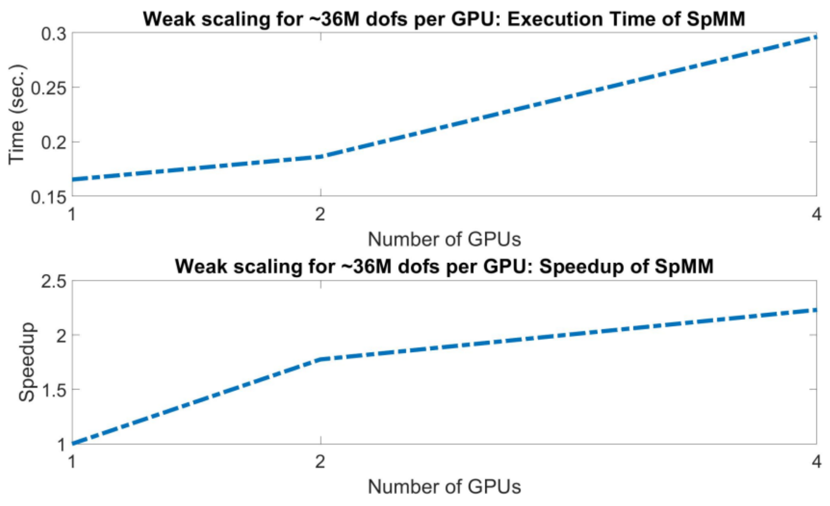


Figure 10 Weak Scalability. SpMM kernel: Execution time (top) and Speedup (bottom)

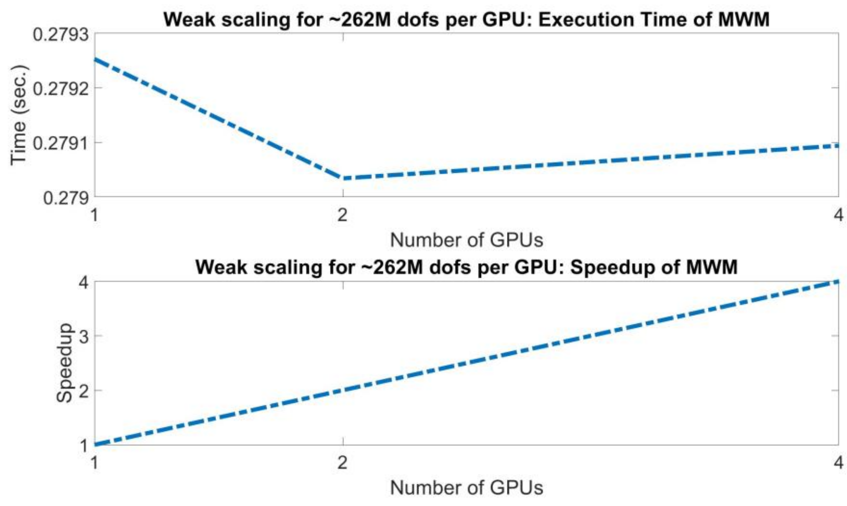


Figure 11 Weak Scalability. MWM kernel: Execution time (top) and Speedup (bottom)

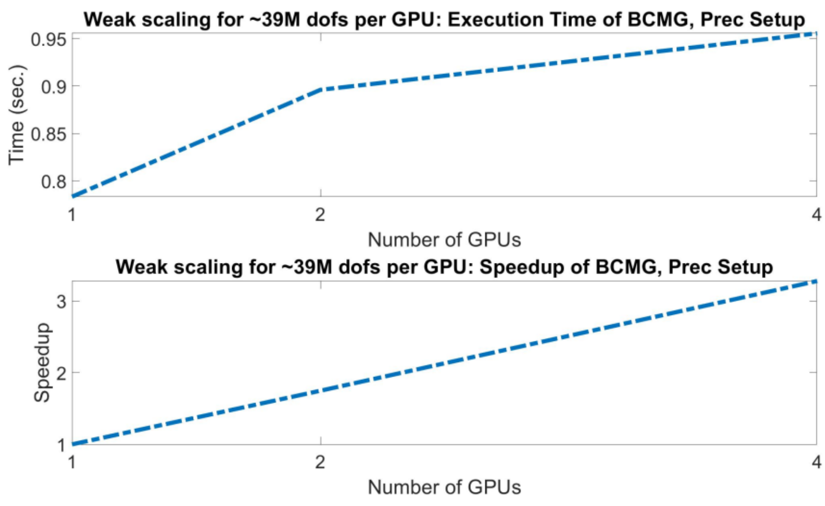


Figure 12 Weak Scalability. BCMG Preconditioner setup: Execution time (top) and Speedup (bottom)

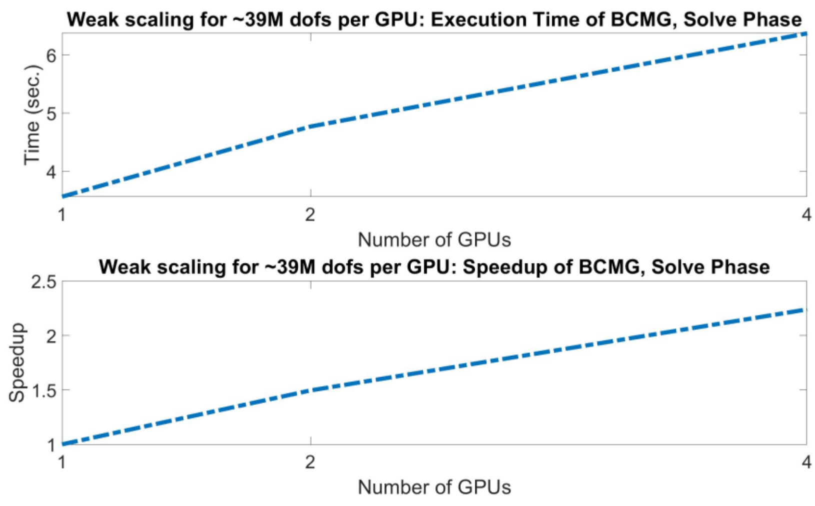


Figure 13 Weak Scalability. BCMG Solve: Execution time (top) and Speedup (bottom)

3.2.1.3 Preliminary measures of Power Consumption on Dibona

In this section we show some preliminary results of power consumption required by our kernels. The measures were obtained by applying the methodology defined by the Textarossa Working Group on Power Measurement and included in deliverable D1.4. In more details, we were able to use the tool developed by INFN relying on the NVML Nvidia library, when our kernels run on 1 GPU at full load. Note that for the BCMG solver, we show in the same picture both the preconditioner setup phase and the solve phase. Furthermore, in order to analyze the possible gain in power consumption, when all the 4 GPUs of the Dibona cluster are used at full load, we did some measures by using the *nvidia-smi* tool, which is able to show counters from all the GPUs available on a single node. Finally, we also analyze the CPU-core power consumption measures obtained by the *likwid* tool accessing the RAPL counters, when our BCMG solve was running, in a weak scalability setting, on 1, 2 and 4 GPUs available on the Dibona node. For the sake of reproducibility, we report in the following the Command Line used for obtaining the measures.

Likwid + INFN tool:

```
      likwid-perfctr  -C  S0:1  -g  ENERGY  -t  1ms  -O  -o  <likwid_output_file>
<application>
```

Likwid + nvidia-smi:

1. Nvidia-smi activation:

```
nvidia-smi dmon -s pucvt -o DT -d 1 -f  <nvidiasmi_output_file>
```

2. Application run (4 GPUs):

```
      mpirun   -np   1   likwid-perfctr   -C   S0:1   -g   ENERGY   -t   1ms   -O   -o
<likwid_output_file>  <application> : -np 1 likwid-perfctr -C S0:2 -g ENERGY -t
1ms -O -o <likwid_output_file>  <application> : -np 1 likwid-perfctr -C S0:3 -g
ENERGY -t 1ms -O -o <likwid_output_file>  <application> : -np 1 likwid-perfctr -C
S0:4 -g ENERGY -t 1ms -O -o <likwid_output_file>  <application>
```

A different <likwid_output_file> is created for each used core (-C S0:X)

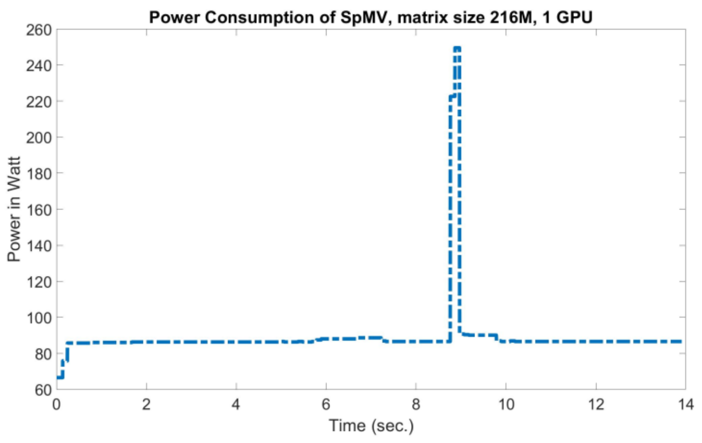


Figure 14 SpMV kernel. Power Measurement on 1 GPU at full load

In Figure 14 we can observe a very sharp phase of increasing power consumption after about 9 sec. from the starting of the SpMV kernel. The measured peak value is about 240 W, while an integral of about 1.2 kW of total power is measured for the global run.

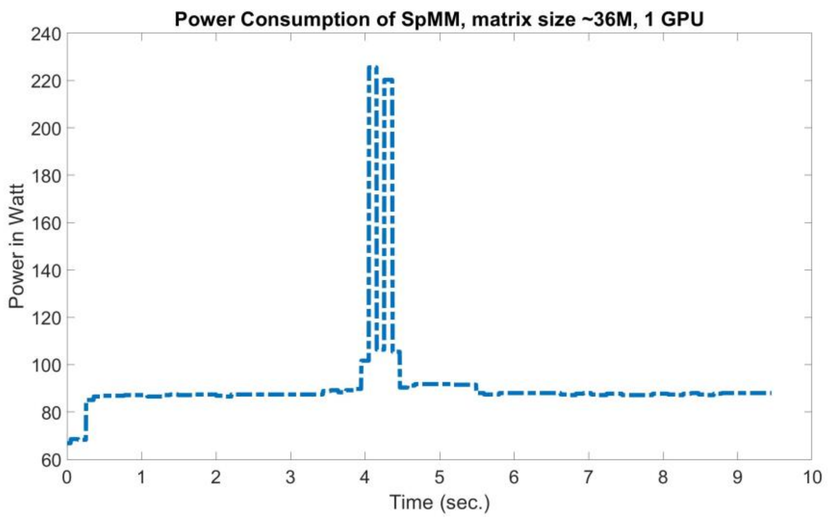


Figure 15 SpMM kernel. Power Measurement on 1 GPU at full load

In Figure 15, we can clearly observe two different increasing phases in the computation, corresponding to the two different computation phases of the SpMM kernel on the GPU. Here we reach two peaks of about 220 W after about 4 sec. from the starting. The integral of the power consumption over the global run is about 861 W.

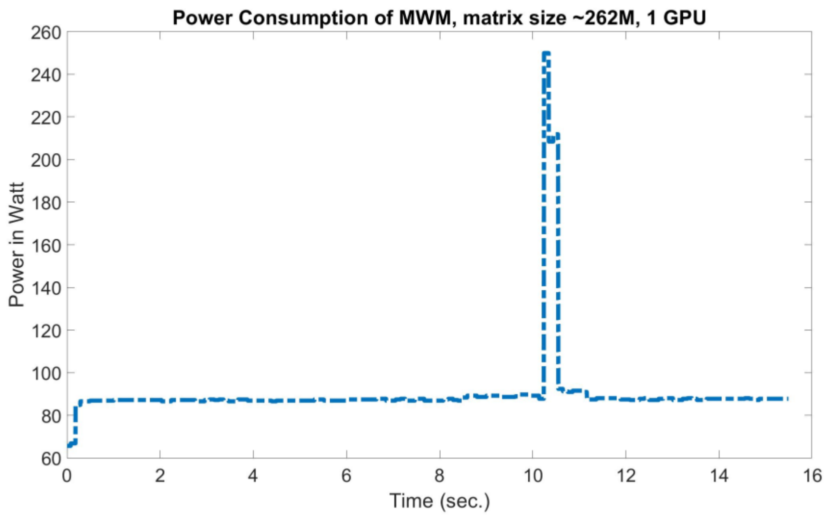


Figure 16 MWM kernel. Power Measurement on 1 GPU at full load

In Figure 16, for the MWM kernel we have a rapid increase after 10 sec. from starting and we see a peak of about 240 W as in the case of SpMV. A global power consumption of 1.4 kW is measured for MWM.

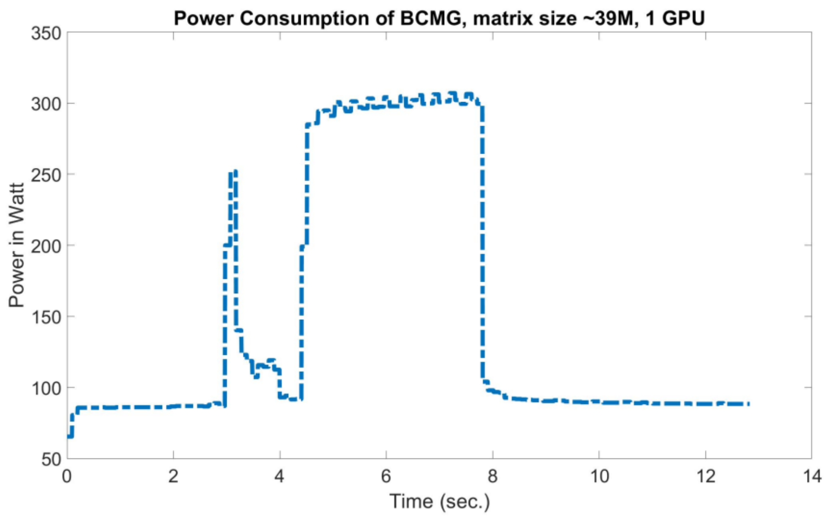


Figure 17 BCMG Preconditioner Setup and Solve. Power Measurement on 1 GPU at full load

In Figure 17 we show power consumption of the BCMG solver. Here we can clearly recognize the two different phases of the computation, that is the preconditioner setup in the interval between 2 and 4 sec. from starting, where a peak of about 250 W is observed, while a longer solve phase in the interval between 4 and 8 sec. is observed, where a power consumption of about 300 W is measured. Here the global run requires about 1.9 kW power consumption.

In Table 3 we summarize Energy KPIs obtained by our kernels, as defined in Table 2, when one GPU is used. In this case, as power consumption value, we consider the integral values of the overall execution period of the kernel.

Kernel name	Dofs/Watt	Iterations/Watt
SpMV	1.75×10^5	Not applicable
SpMM	4.17×10^4	Not applicable
MWM	1.88×10^5	Not applicable
BCMG	2.07×10^4	0.02

Table 3 Energy KPIs on 1 GPU

In Figure 18 we show the measures of power consumption when the overall BCMG solver is run on an increasing number of GPUs, while the problem size linearly increases with the number of GPUs, as in the weak scalability setting previously discussed. We can observe that when 4 GPUs are involved in the computation, a smaller peak of about 240 W is observed for all the GPUs, while larger peaks of about 280 W and 300 W are observed when 2 GPUs and 1 GPU are used. This result on 1 GPU is in a perfect agreement with that already showed in Figure 17. If we compute the integral of the power consumption of all the 4 GPUs in the 3 different configurations, i.e., when the solver run on 1, 2 and 4 GPUs respectively, for solving systems with 39M, 78M and 156M unknowns, respectively, we measure a global power consumption of about 2.3 kW, 2.2 kW and 2.5 kW, respectively. This result demonstrates that, if we are able to use in an efficient way all the devices integrated on the single node at full load, we can solve larger problems by an almost constant energy consumption, leading to achieve energy scalability in addition to performance scalability of the computation.

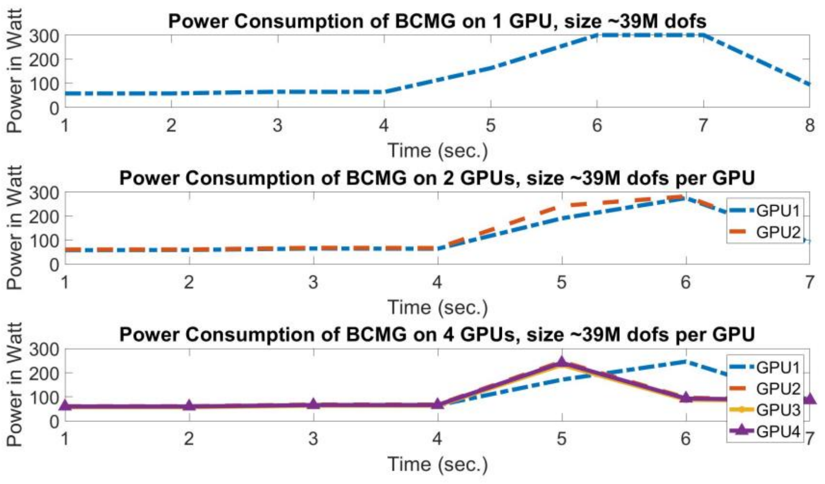


Figure 18 BCMG Preconditioner Setup and Solve on multiple GPUs at full load

In Table 4 we summarize Energy KPIs obtained by BCMG when more than 1 GPU are used in a weak scaling setting. As we can see, running our solver on more than 1 GPUs increases the energy efficiency, indeed, we can deal with a larger number of Dofs and solve iterations at the same energy power.

	Dofs/Watt	Iterations/Watt
BCMG on 2 GPUs	3.54×10^4	0.03
BCMG on 4 GPUs	6.39×10^4	0.03

Table 4 Energy KPIs when multiple GPUs are used in a weak scaling setting

For the sake of completeness, in Figure 19 we show the power consumption measures obtained by the likwid tool when BCMG is running on 1, 2 and 4 GPUs. The very small and almost stable power consumption, during all the computation, demonstrates that in all the cases, CPU cores are not involved in a significant way in the computation.

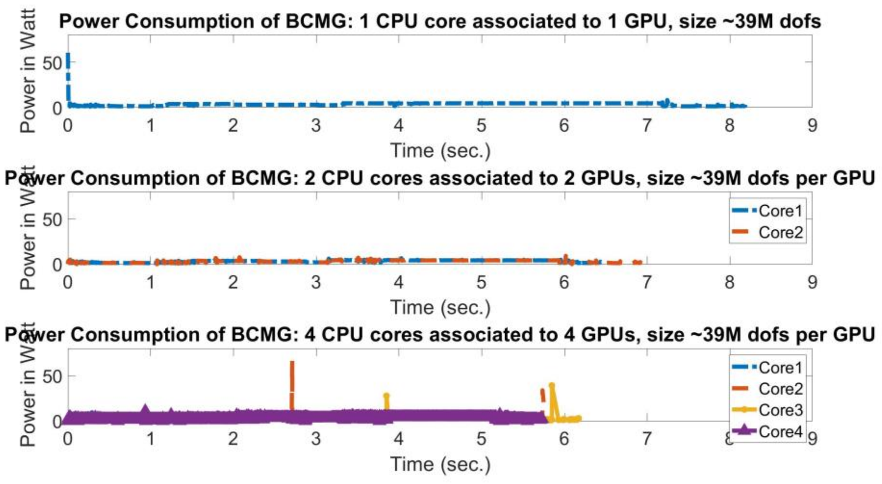


Figure 19 BCMG Preconditioner Setup and Solve on multiple GPUs by Likwid CPU counters

3.2.2 Weak Scalability results of BCMG and Comparison with the State of the Art on the Piz Daint Supercomputers

In this section, we analyze the scalability potential of our BCMG solver when the number of GPUs largely increases; the fixed matrix size per node is equal to 130^3=~2.2M Dofs, going from 1 to 100 nodes. Therefore, we solve problems up to 220M unknowns. We analyze the performance of the linear solver looking also at the number of iterations of the PCG, in order to analyze the algorithmic scalability, i.e., the potential to have an almost constant or slowly increasing number of iterations for increasing number of unknowns and computing resources. The execution time to solve the system and the execution time per each PCG iteration are also discussed to characterize the application phase from the viewpoint of the implementation scalability. Therefore, in this case, in the following Figures, we report all KPIs which characterize a linear solver.

Our hybrid BCMG, is compared with the hybrid version of Nvidia AmgX [4,5]. 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 the input configuration which defines AMG hierarchies based on a similar coarsening approach and having complexities comparable with our preconditioner.

In Figure 20, we show number of iterations of BCMG versus AMGX. We can see that in all cases the number of iterations required by AMGX in the solve phase is always higher than that of BCMG. After an initial increase for both the solvers, they have a similar more stable behavior, but the increase in the number of iterations for AMGX is ~50% going from 1 to 100 nodes, whereas the increase for BCMG is ~36%, showing that BCMG has better algorithmic scalability.

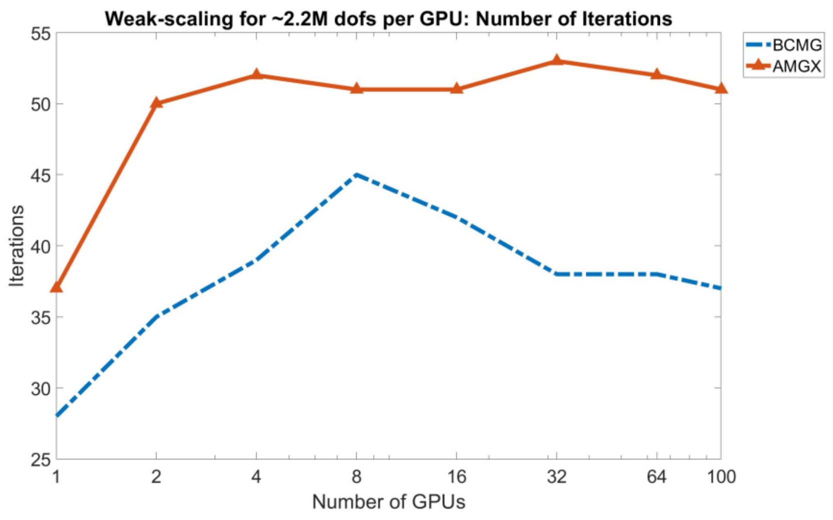


Figure 20 Weak Scalability. Number of iterations of BCMG vs AMGX

This better quality of our preconditioner is confirmed by the solve time (see Figure 21).

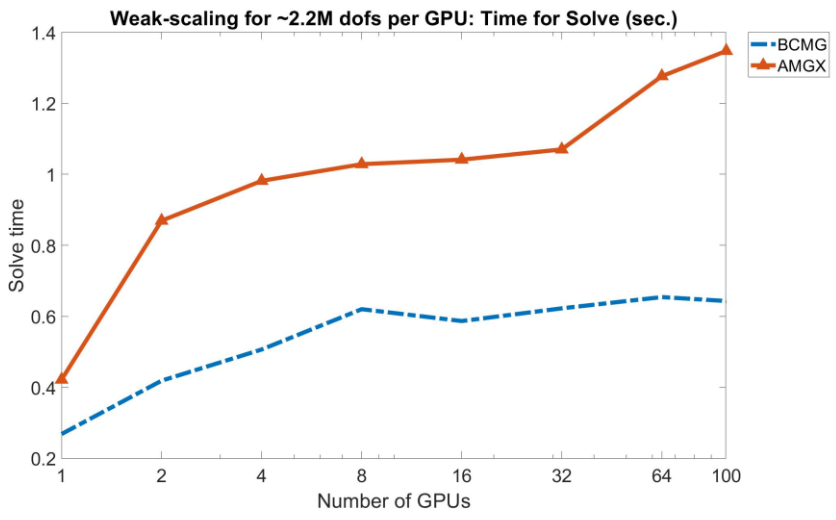


Figure 21 Weak Scalability. Time to solution of BCMG vs AMGX

BCMG solve times are always significantly smaller than that of AMGX. In many cases AMGX requires a solve time that is double than that of BCMG. Finally, if we look at the time per iteration, measured as the ratio between the time to solution and the number of iterations needed to converge (see Figure 22), we see that BCMG always has smaller time than AMGX. On the other hand, BCMG also shows a smaller increase ratio for increasing number of nodes, showing that all the computational kernels in the application phase of the preconditioner are efficiently implemented.

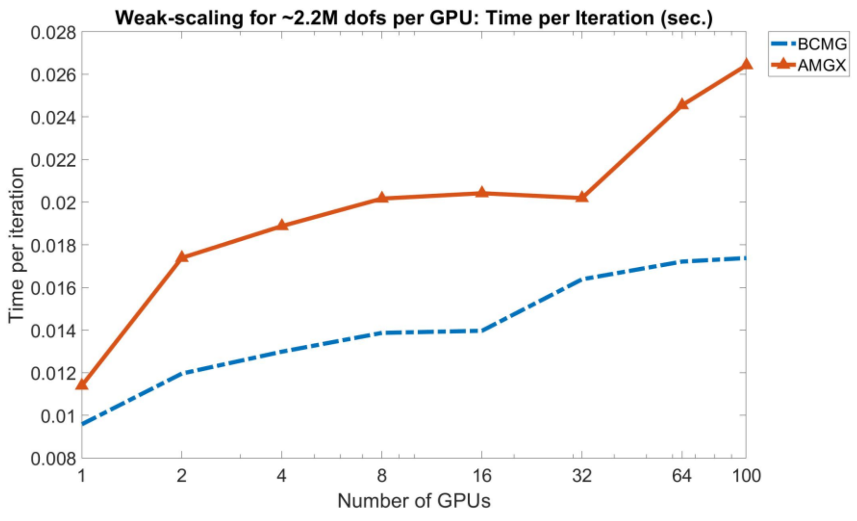


Figure 22 Weak Scalability. Time per iteration of BCMG vs AMGX

3.3 RTM – FRAUNHOFER

In this project a simple implementation of the 3D isotropic RTM Kernel has been extended using different formats of reduced precision to compress the domain and the model to save memory bandwidth. The kernels are still computed in float32, but the domains and velocity model are kept in the compression format. Conversions are done within the kernel to switch between float 32 bit and the compression format. The imaging condition and aggregation is done also in float 32bit. The domains are converted to 32 bit to do so. Implementation D is special because not only the domains and model are kept in the compression format, but also the computational steps in the kernel are done in the compression format. However, even in this implementation the imaging condition is calculated in float 32bit format. The formats are detailed in Table 5.

Label	Description
A	10 bit floating point, implemented by GNU MPFR library, emulating float 16 bit
B	7 bit floating point, implemented by GNU MPFR library, emulating bfloat 16 bit
C	Posit 16 bit, one exponent bit, implemented by SoftPosit library
D	Posit 16 bit, one exponent bit, implemented by SoftPosit library, including kernel computing
E	ZFP array1f 1D, 16 bit per element, implemented by ZFP library
F	ZFP array3f 3D, 16 bit per element, implemented by ZFP library

Table 5 List of reduced precision formats.

Please note that in case A and B the exponent is unlimited, other than in floating point and bfloat formats.
Further the RTM implementation has been extended to calculate and to document the total energy of the source volume over time in form as the sum of squares on the domain. These outputs are used to verify the numerical stability.

3.3.1 Results

The following table depicts the results of a single shot calculated in 3D. The model has two horizontal layers of constant velocity. The images of the two versions are compared by voxel vise numerical difference followed by a maximum norm. Perfect outcome would be zero. See the results in Table 6.

	Floating Point Format	Maximum norm	Maximum norm of difference to Floating Point 32 Bit
	Float 32 bit	3857	0
A	Float 16 bit	3683	460
B	Bfloat 16 bit	691520	691616
C	Posit 16 bit	3687	647
D	Posit 16 bit + kernel	4270	2356
E	ZFP, 1D	3869	157
F	ZFP, 3D	3863	21

Table 6 Comparison of different compression formats.

Figure 23 plots the ratio of the fourth column and the third column. The bfloat format does not give useful results. Format A and C end up with roughly 10% deviation. Calculating the kernel in posit 16 bit ends up with 60%. ZFP 1D yields 4% deviation while ZFP 3D yields below 1% deviation.

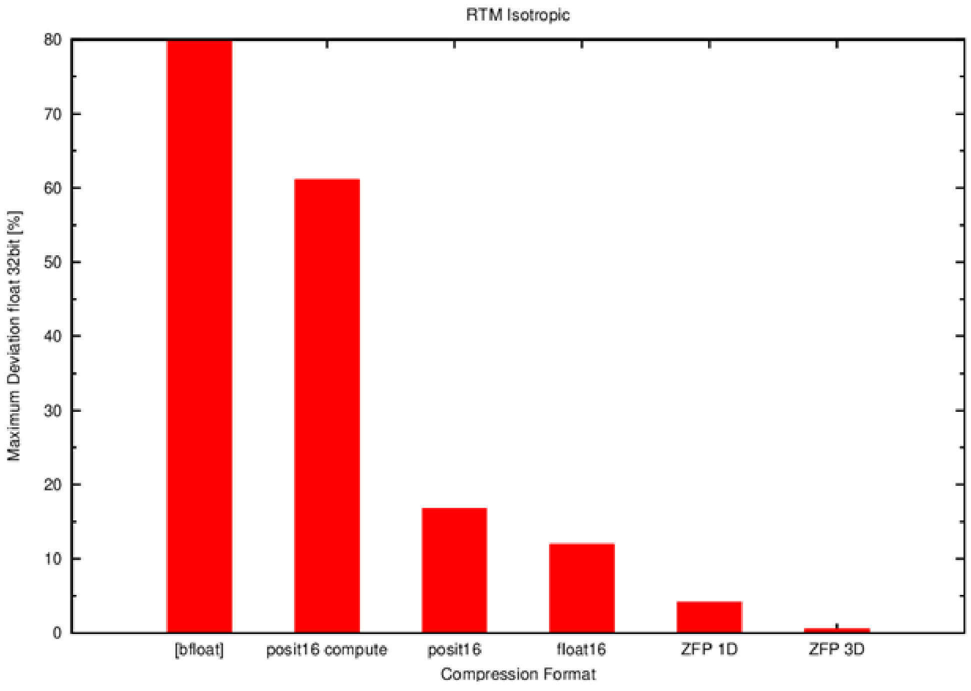


Figure 23 Plot of relative maximum norm of difference between compression format and float 32bit. Normalized on float 32 bit. Bfloat exceeds 100% deviation.

However, the numerical differences only give a rough idea of image quality. Figure 24 presents slices perpendicular to the depth axis for each implementation. The resulting slice and the reference slice are plotted side by side. The third column depicts the difference of the previous columns.
General criteria of quality are qualitative similarity and the absence of structure in the difference plot.
Implementation A shows very few visible differences but the difference plot shows an unsymmetric and unphysical structure. Implementation B shows unusable results. Implementation C shows few differences and a very symmetric difference plot. Implementation D shows significant and also unsymmetric differences. The same goes for the difference plot. Implementations E and F both show very good similarity but very unsymmetric and unphysical difference plots.

To verify stability the total energy of the source volume is plotted over time for different time steps, namely 2.1ms, 2.2ms and 2.3ms. The velocity model is constant. Figure 25 depicts a slice perpendicular to the depth axis of the source volume. In Figure 26 all implementations show stability up to 2.2ms and unstability for 2.3ms. The only exception is implementation D which is unstable at 2.2ms but stable at 2.1ms. So the usage of compression formats has no significant influence on stability. All implementations except D have less than 5% loss in stable time steps.

A second and more challenging stability test uses a velocity model filled with random numbers equally distributed between 1250m/s and 2750m/s. Figure 27 on the left side proves stability for float 32 bit implementation up to 1.9ms and instability at 2.0ms timestep. All the compression implementations show stability at a timestep of 1.9ms as depicted on the right side. So even using a very rough velocity model the stable time step is kept within a 5% range around the floating-point implementation.

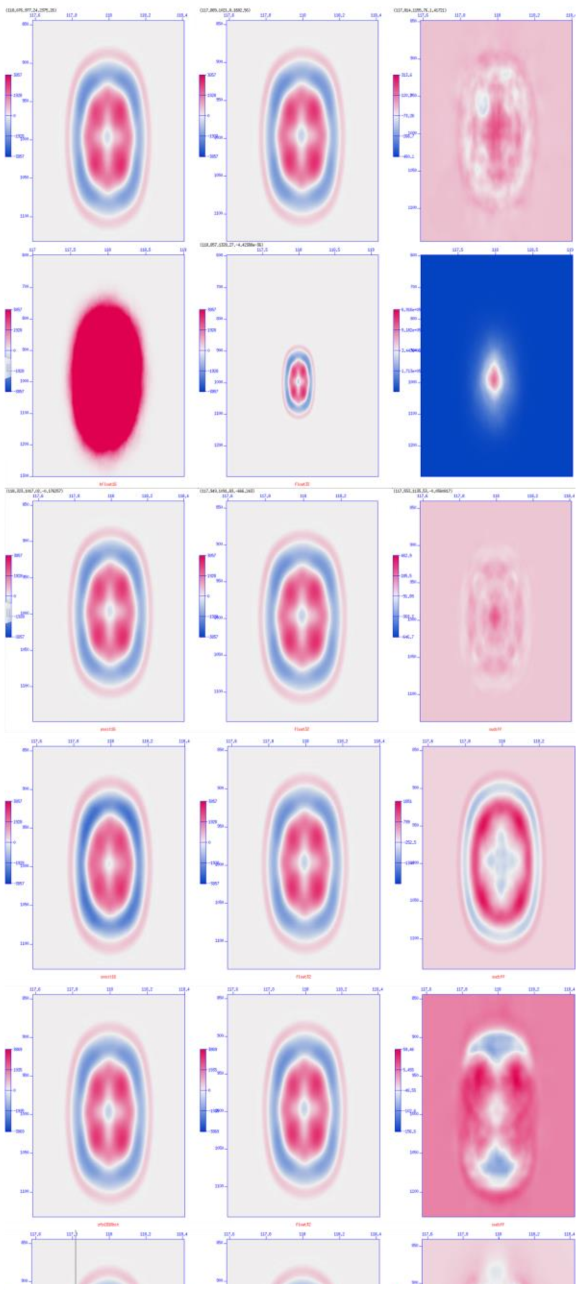


Figure 24 Cut perpendicular the depth axis for all the compression formats

First column compression format, second column reference Float 32bit, third column difference between first column and second column. Compression formats (top to bottom): A: Float 16 bit, B: Bfloat 16 bit, C: Posit 16 bit with 1 exponent, D: Posit 16bit storage and computation, E: ZFP 1D 16 bit, F: ZFP 3D 16bit

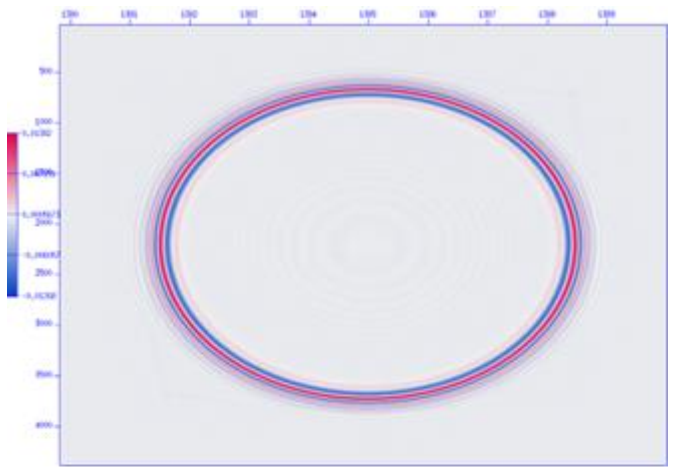


Figure 25 Cut of source volume perpendicular to x-axis

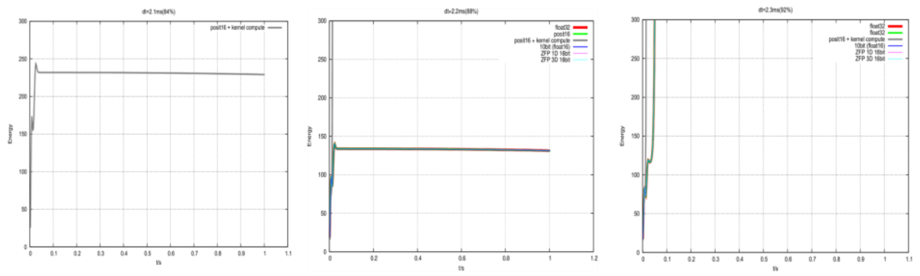
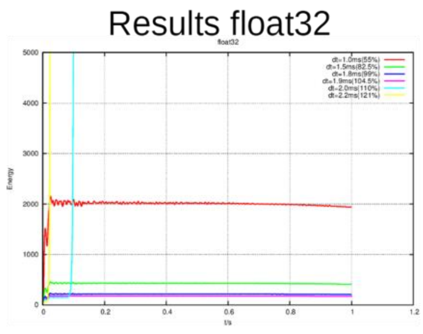


Figure 26 Total energy of source over time for different time step size Constant velocity model. Left: dt=2.1ms, Middle: dt=2.2ms, Right. 2.3ms.



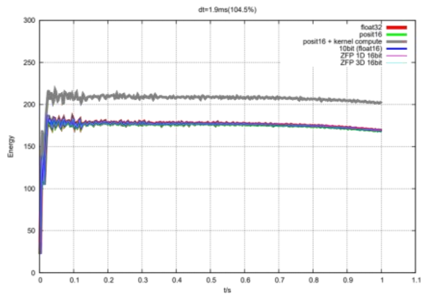


Figure 27 Total energy of source over time for different time step size Non constant velocity model. Left: Different time steps for float 32 bit. Right: All the compression formats for dt=1.9ms.

3.4 HEP

As we described in Deliverable 6.1, it is important to execute High-Energy Physics (HEP) code on heterogeneous architectures due to the evolution of the computing solution offered by accelerator technologies.

For the TEXTAROSSA project we have worked on two different applications: Pixeltrack [6], a track reconstruction algorithm for the CMS detector, and CLUE [7], a cluster algorithm for high-granularity calorimeters. Both applications are mainly developed by the CERN Patatrack team.

Our first task was to rewrite the abovementioned applications on top of a portability layer, with the purpose to obtain a single source code per application that can be run in parallel on multiple heterogeneous backends; in fact, it is not affordable to redesign and reimplement the algorithms for each different architecture of every vendor. The next goal has been the evaluation of the application in terms of:

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

Our main interest is the comparison of those two metrics between the heterogeneous version of the application and the corresponding serial one.

Our initial plan was to use SYCL [8], the standard abstraction layer based on ISO C++. More specifically, we planned to use the SYCL implementation coming with Intel oneAPI, since it is the most complete implementation of the standard. We have obtained a SYCL version of CLUE and a SYCL version of Pixeltrack correctly working on CPUs, Intel GPUs, and Intel FPGAs. We have also tried to port the heterogeneous code on other architectures of different vendors, such as NVIDIA GPUs and AMD GPUs, but we have not obtained an executable code yet. The results we achieved have required a lot of effort because the compiler is not stable yet: Intel is still working on it with the goal of extending the supported hardware.

Some results of this work were presented at the 21st International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2022), with a poster called "Experience in SYCL/oneAPI for event reconstruction at the CMS experiment" [9]. As shown in the poster, the performance achieved with the SYCL version of CLUE on an NVIDIA Tesla T4 is promising and in line with other technologies.

For the purpose of the TEXTAROSSA project, we conclude that SYCL is a very promising abstraction layer but its implementation in oneAPI is not stable enough and not ready yet to be used for our purposes, that is, run the applications used to acquire and elaborate the data collected from the HEP detectors on different heterogeneous hardware. For that reason, we have decided to collect the data for the TEXTAROSSA project with a different abstraction layer, Alpaka [10]. From the documentation: "The Alpaka library is a header-only C++14 abstraction library for accelerator development. Its aim is to provide performance portability across accelerators through the abstraction (not hiding!) of the underlying levels of parallelism". Although coding with Alpaka is more complicated than coding with SYCL, this library is supported for a wider range of devices of different vendors than oneAPI, making it more appropriate for our goal.

Parallel to SYCL, we have implemented the Alpaka version of Pixeltrack and the Alpaka version of CLUE. The same version of the source code can run on different devices without any modification. We have decided to perform all the measurements with that version of our applications. The architecture we used for the TEXTAROSSA tests was the IDV-A. We use both the CPU and the GPUs that this architecture provides.

3.4.1 Tests on CPU

On the CPU we are interested in demonstrating that our code scales with the number of resources we use for the execution; we would also investigate how the energy efficiency changes with respect to the same parameter.

We ran both our applications increasing the number of cores and the number of CPU threads in order to execute them with one thread per core. For the power consumption measurements, we use the *likwid-perfctr* tool; it also helps us in specifying the correct number of cores we want to use. 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 3-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 3 --numberOfThreads <threads>
```

The results we achieved are reported in the section below. Table 7 and Table 8 showed the measures of throughput (events/second), Power (W) end Energy Efficiency (events/J) for the CLUE application and for the Pixeltrack application when they run on CPU, scaling the number of cores.

CLUE on CPU				
cores	thread	Throughput (events/second)	Power (W)	Energy efficiency (events/J)
1	1	4.33347	81.140	0.053
2	2	8.5591	84.338	0.102
4	4	16.8088	91.589	0.184
8	8	33.5036	105.825	0.317
16	16	66.2713	134.687	0.492
32	32	118.869	173.500	0.685
48	48	156.511	191.146	0.819
64	64	216.123	321.966	0.671
96	96	271.130	235.522	1.151

Table 7 Scaling of KPIs for the CLUE application on CPU

Pixeltrack on CPU				
cores	thread	Throughput (events/second)	Power (W)	Energy efficiency (events/J)
1	1	30.8127	81.944	0.376
2	2	59.9623	86.431	0.694
6	6	180.194	103.412	1.742
12	12	360.465	128.886	2.797
24	24	726.897	181.967	3.995
36 on 1 socket	36	860.592	198.681	4.332
36 on 2 sockets	36	1068.88	305.188	3.502
48	48	1418.03	354.890	3.996
72	72	1636.91	390.303	4.194
96	96	1756.96	233.529	7.524

Table 8 Scaling of KPIs for the Pixeltrack application on CPU

Figure 28 and Figure 30 show the scaling of throughput for the two applications; Figure 29 and Figure 31 represent the correlation between the cores and the energy efficiency. Note that in the figures the data are normalized with respect to the values obtained with 1 thread on 1 core.

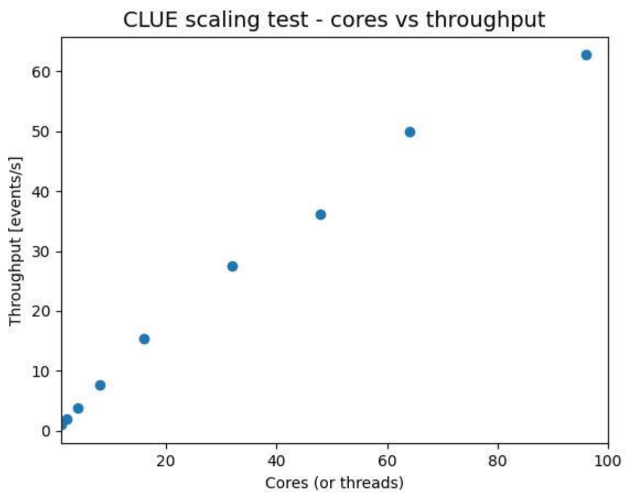


Figure 28 Scaling of throughput with the number of used cores for the CLUE application on CPU

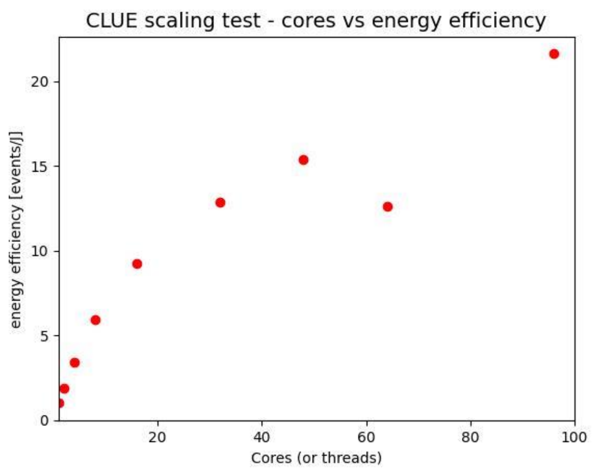


Figure 29 Scaling of energy efficiency with the number of used cores for the CLUE application on CPU

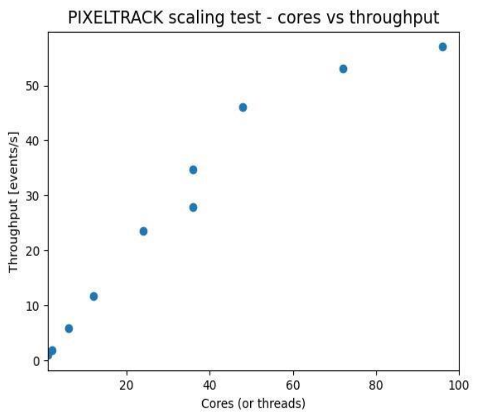


Figure 30 Scaling of throughput with the number of used cores for the Pixeltrack application on CPU

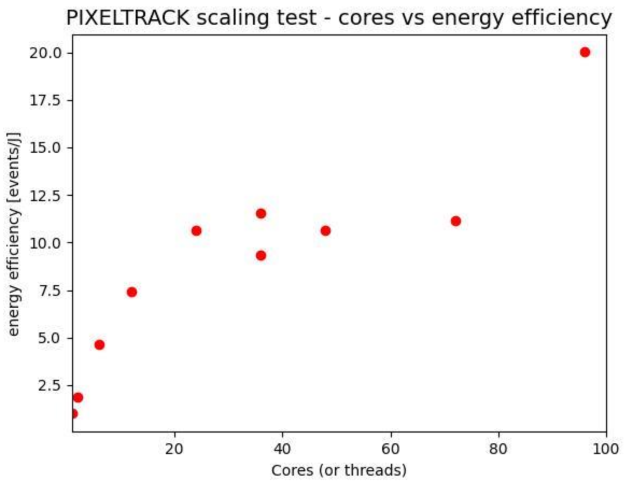


Figure 31 Scaling of energy efficiency with the number of used cores for the Pixeltrack application on CPU

Figure 28 and Figure 30 show that the performance grows linearly when we run one thread per core using one socket. This trend remains constant until we use entirely the first processor.

When we run more threads per core using two sockets, the performance continues to increase but the increment starts to be slower. The same behavior is evident also with two processors (more than 48 cores).

Regarding Figure 29 and Figure 31, we notice that there is an increment of the energy efficiency, even if it is not linear, until we use 48 threads on 48 logical cores. After that point, the efficiency gets worse; this is due to the fact that scaling beyond that number we start using two sockets and we incur in the overhead of using the second CPU but at a fraction of its processing capabilities. The penalty is evident in Figure 31 where we report two values for the energy efficiency for 36 logical cores: the higher value corresponds to using just one socket, the lower when using 24 cores on one socket and 12 on the other one.

3.4.2 Tests on GPU

On the GPU we are interested in demonstrating that our code scales with the number of GPUs we use; we would also investigate how the energy efficiency changes with respect to the same parameter.

We decided to fix to 12 the number of CPU threads per GPU, mapping each of them on one different core. In this way, with 4 GPUs all the 48 physical cores are busy. The number of CPU threads, as in the previous case, could be indicated in the command we use to run the application. Also, the mapping between GPU, CPU threads and cores could be set in the same command. We executed all our tests for 2 minutes. Summing all these considerations, we report for example the command we use to run 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 9 and Table 10 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			
gpu	total throughput (events/second)	power (W)	power efficiency (events/J)
1	1485.64	183.333	8.103
2	2843.05	362.708	7.839
3	4248.17	541.583	7.844
4	5680.66	729.708	7.785

Table 9 Scaling of KPIs for the CLUE application on GPU

Pixeltrack on GPU			
gpu	total throughput (events/second)	power (W)	power efficiency (events/J)
1	2174.26	150.375	14.459
2	4386.89	303.261	14.466
3	6515.27	450.750	14.454
4	8711.73	606.458	14.365

Table 10 Scaling of KPIs for the Pixeltrack application on GPU

Figure 32 and Figure 34 and show the scaling of throughput for the two applications when we increment the number of GPUs; Figure 33 and Figure 35 represent the correlation between the GPU number and the energy efficiency. Note that in the figures the data are normalized with respect to the values obtained with 1 GPU.

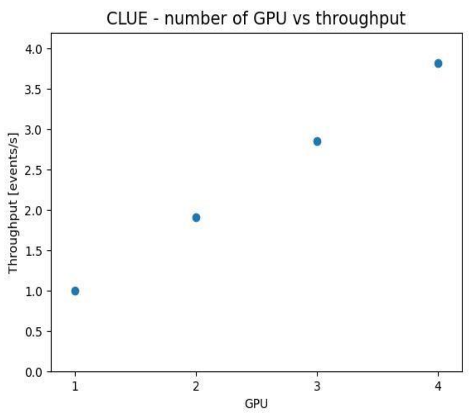


Figure 32 Scaling of throughput with the number of used GPUs for the CLUE application

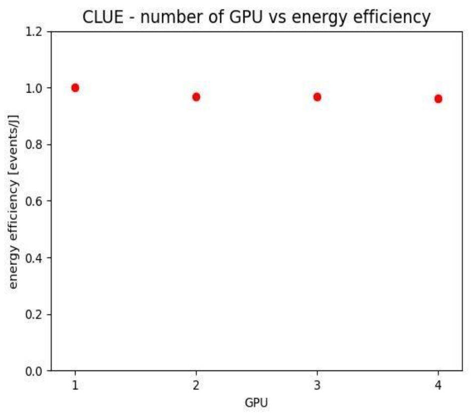


Figure 33 Scaling of energy efficiency with the number of used GPUs for the CLUE application

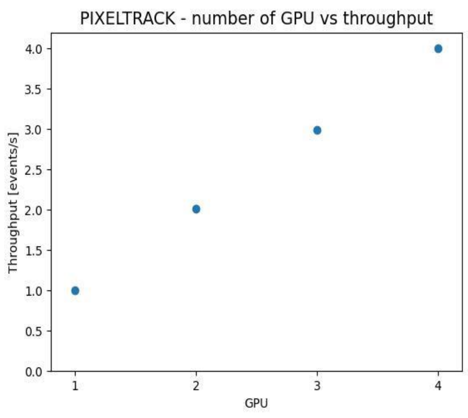


Figure 34 Scaling of throughput with the number of used GPUs for the Pixeltrack application

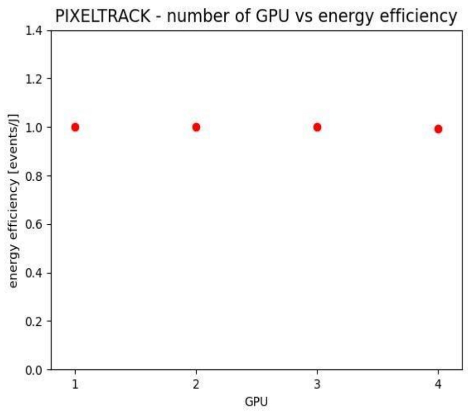


Figure 35 Scaling of energy efficiency with the number of used GPUs for the Pixeltrack application

From the Figure 32 and Figure 34 we notice that, as we expected, both the applications scale linearly with the number of GPUs: every GPU has the same throughput, so doubling the GPUs doubles the total throughput. Regarding energy efficiency, we notice that it remains almost constant. That behavior shows that, even if we introduce hardware, it allows us to increase the throughput, but the number of events processed per Joule remains constant.

3.5 NEST-GPU – INFN

The neural simulation application is considered production-ready and is being used in recently published papers [11], but it is being modified in order to be in compliance with the rules of the NEST application building framework; this means that the activities currently being performed regarding Textarossa are a number of syntactical adjustments in the scripts that complement the application in the project set of benchmarks in order to keep them working as when the application was still NeuronGPU.

The first trials were performed with a setup that simulates a section of mouse cortex with neurophysiological details inferred from experiments. This was initially run on the CPU-only version of the NEST simulator on the Ampere Altra platform made available by E4; the tests take as reference a schedule made of 1000 simulated milliseconds which are discarded (to let the network reach a reasonably steady and homogeneous state among all cortical columns), then a 1000-2000-4000 timeline, in order to check that no large differences in runtime among different MPI processes occur (so that load is evenly distributed) and the runtime scales with the simulated time. This network appears to have a level of average activity of 5M spikes per simulated second, so we use the number of Spike Updates per Joule (SUs/J) in the simulated time interval as another performance index.

The scaling results on the Ampere Altra (for the final leg of simulation, 4000ms) are in the Table 11 Runtimes for the last leg (4000ms) of simulation on different process/cores layouts on Ampere Altra..

32 cores tot		64 cores tot		128 cores tot	256 cores tot
1 MPI prc 32 cores/prc	2 MPI prc 32 cores/prc	4 MPI prc 16 cores/prc	8 MPI prc 8 cores/prc	16 MPI prc 8 cores/prc	32 MPI prc 8 cores/prc
156.91s	126.83s	107.43s	88.86s	78.04s	138.83s

Table 11 Runtimes for the last leg (4000ms) of simulation on different process/cores layouts on Ampere Altra.

We can see that some consideration must be taken when choosing how to distribute the available cores for the run – the application can take significantly longer when the same number of cores (64) is employed (from 126.83s when using 2 MPI processes and 32 OpenMP threads per process against 88.04s for the same cores allocated in 8 MPI processes with 8 OpenMP threads per process) or the scaling limit is reached when trying to push to 256 cores (138.83s) makes the application actually take longer than when using just 128 cores (78.04s).
The significant KPI here is the simulated seconds per second, which on this platform in the best case (with 128 cores) is 4000/78.04 = 51.26 simulated milliseconds per second.

Table 12 presents the power readings on the Ampere Altra; given that there is no support for the processor in LIKWID, the sensors utility was used with 0.7s sampling interval between calls over the entire 8000 simulated milliseconds timeline. In this table we report the energy spent (in kJ) during this simulation phase, the actual runtime and the average wattage for every configuration.

32 cores tot		64 cores tot		128 cores tot	256 cores tot
1 MPI prc 32 cores/prc	2 MPI prc 32 cores/prc	4 MPI prc 16 cores/prc	8 MPI prc 8 cores/prc	16 MPI prc 8 cores/prc	32 MPI prc 8 cores/prc

47.9kJ	32.7kJ	37.6kJ	33.0kJ	35.9kJ	62.9kJ
461s	243s	253s	191s	166s	283s
104W	135W	149W	173W	216W	222W

Table 12 Power, runtime and wattage per configuration over total simulation (8000ms) on Ampera Altra.

Figure 36 (time in seconds on the X-axis, power in Watts on the Y-axis as read by the sensors command and summed over the reading for the two CPU sockets) is to be read as follows: for the different processes/cores arrangements, the application has a brief startup phase (the first few seconds-long pedestal), then a longer (a few hundred seconds, depending on how many OpenMP cores were assigned to the process), higher plateau where some setup operations for the cortical areas are performed, then the actual simulation, with the mentioned 1000ms (discarded) – 1000ms – 2000ms – 4000ms, for a total of 8000 simulated milliseconds (the plateau on the right).
It can be seen this second plateau to become shorter and higher as the cores are increased; an interesting and unexpected finding for this platform is that while the highest power consumption is expected to occur in this phase, this does NOT happen when pushing all 256 cores, when the most power is drawn in the setup phase (the first plateau for the yellow line is higher than the second one).
It is also worth mentioning that the actual minimum for the energy at 32.7kJ does not appear to be achieved by the configuration taking the shortest time (the 128 total cores configuration) but by an intermediate one (the 2 MPI processes/32 cores per process, 64 total cores configuration). Given the estimated level of activity of 5M spikes per simulated second, another KPI is 5M spikes per simulated second * 8 simulated seconds / 32.7kJ = 1223 SUP/J.

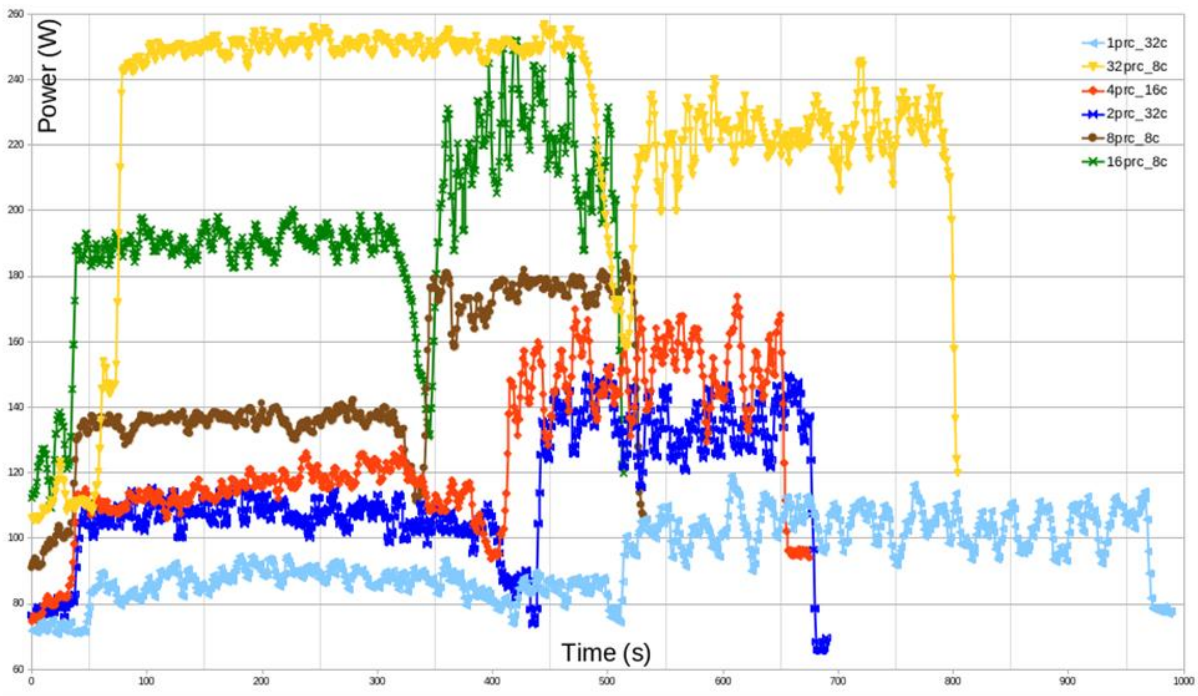


Figure 36 Power profile for 8000 simulated milliseconds in different process/cores layouts on Ampere Altra.

The very same trials were performed on the EPYC CPUs of the Textarossa partition on the Dibona cluster by ATOS; the scaling results (for the 4000ms leg of the schedule) are reported in Table 13.

12 cores tot	24 cores tot		48 cores tot	
2 MPI prc 6 cores/prc	2 MPI prc 12 cores/prc	4 MPI prc 6 cores/prc	2 MPI prc 24 cores/prc	4 MPI prc 12 cores/prc
376.61s	192.67s	192.51s	104.98s	104.13s

Table 13 Runtimes for the last leg (4000ms) of simulation on different process/cores layouts on Dibona EPYC.

Here we see that the runtimes spread times for different configurations of processes and threads are much smaller while the KPI of simulated milliseconds per second is at best 4000/104.13 = 38.41.

Being a standard x86_64 platform, the Dibona EPYC is therefore supported by LIKWID utility; the power readings tabulated below were taken via LIKWID with a 0.5s sampling interval for the entirety of the 8000ms schedule, presented in Table 14.

12 cores tot	24 cores tot		48 cores tot	
2 MPI prc 6 cores/prc	2 MPI prc 12 cores/prc	4 MPI prc 6 cores/prc	2 MPI prc 24 cores/prc	4 MPI prc 12 cores/prc
150.6kJ	95.4kJ	94.9kJ	72.7kJ	72.1kJ
755s	392s	390s	214s	213s
200W	244W	243W	340W	338W

Table 14 Power, runtime and wattage per configuration over total simulation (8000ms) on Dibona EPYC.

Here the situation is different compared to the Ampere Altra: besides cutting the runtime, using more cores of course increases the wattage but not as much as compared to the Ampere Altra, so that the least energy here is drawn for the configuration with most cores and the shortest runtime. With the estimated level of activity of 5M spikes per simulated second, the relevant KPI is at best 5M spikes per simulated second * 8 simulated seconds / 72.1kJ = 554 SUP/J.

As for the Ampere Altra (the colors in the plot distinguish the different configurations described in Table 14), Figure 37 (time in seconds on the X-axis, power in Watts on the Y-axis as reported by LIKWID and summed over the reading for the two CPU sockets) gives an idea of the power profile. The setup phase (the plateau on the left) increases in power consumption adding more cores without being significantly shortened while the simulation phase scales very well (halving the runtime when doubling the cores) while wattage ramps up from 200W to 338W.

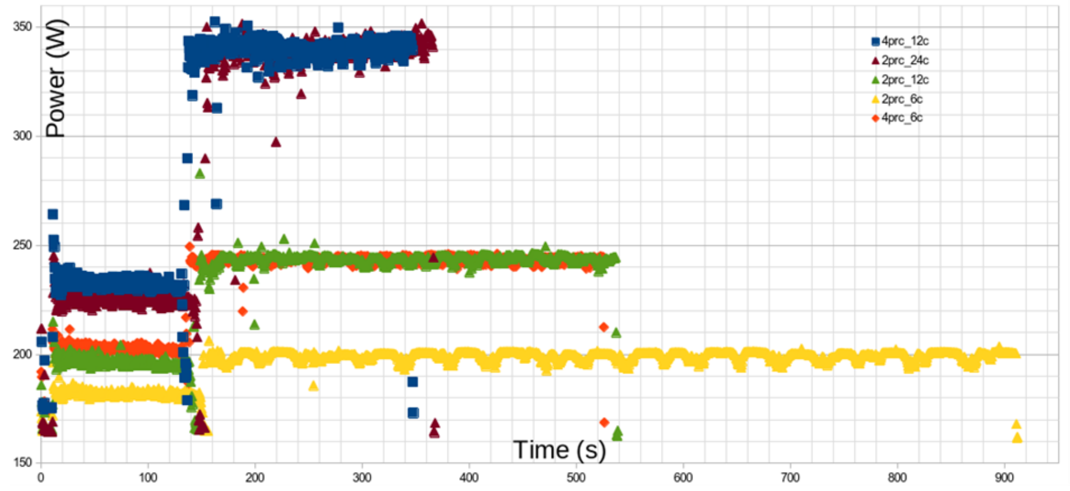


Figure 37 Power profile for 8000 simulated milliseconds in different process/cores layouts on Dibona EPYC.

The actual GPU version of the NEST application was run on a platform sibling of the Dibona EPYC and equipped with the same GPU, an NVIDIA A100 that we had more handily available for testing; we do not expect significant changes to the results.
Performing a scaling test for the NEST-GPU application requires a multi-GPU and multi-node setup that we did not have available; at the same time, the testing script is being reworked to operate in such an environment. For the time being we tried the same test described above in a single GPU configuration test in order to gauge the power consumption while developing the GPowerU tool and have a ballpark figure of the runtimes.
Figure 38 shows the results: it is the output of the GPowerU tool (time in seconds on the X-axis, power in Watts on the Y-axis, sampling period of 0.01s) on a run of 10000 simulated milliseconds executed on an NVIDIA A100 with the setup phase cut from the plot.
The actual duration of the simulation is 33.1s with an average wattage of 118W and a total energy draw of a little less than 4kJ; the derived KPIs result 10000/33.1 = 302 simulated milliseconds per second and 5M spikes per simulated second * 10 simulated seconds / 4kJ = 12500 SUP/J.

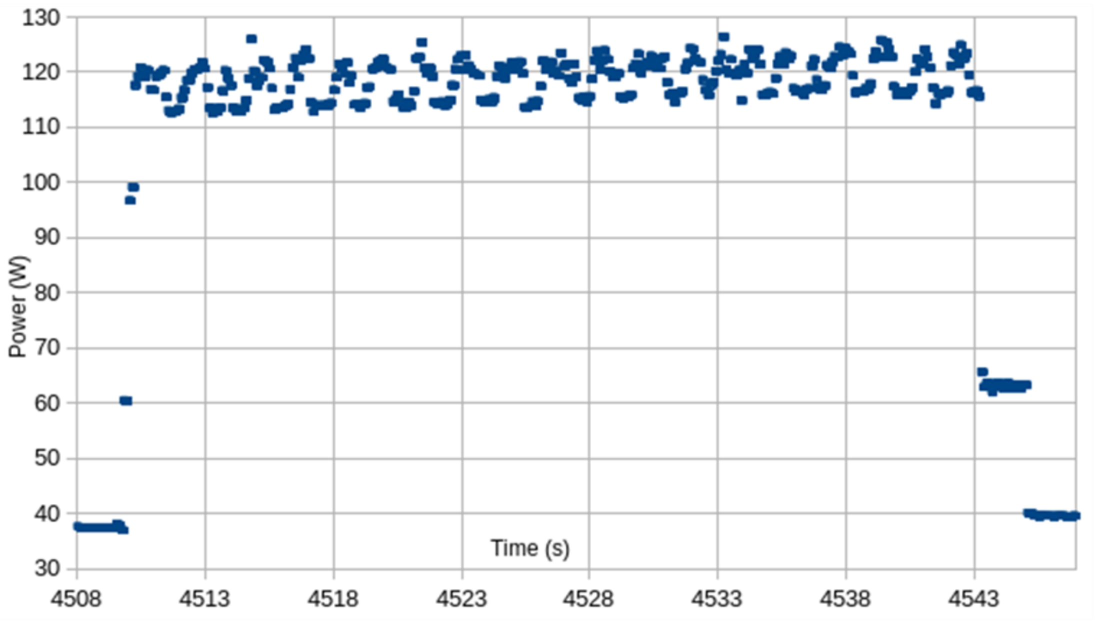


Figure 38 Power profile for 10000 simulated milliseconds on NVIDIA A100 (the first ~4500s of runtime are for the setup phase where the GPU is unused and were cut from the plot).

3.6 RAIDER – INFN

RAIDER is a high throughput online streaming processing application implemented on FPGA with the APEIRON framework and belongs to the HPDA domain. 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 at a rate of about 10 MHz, using neural networks.
In this preliminary version of the RAIDER application, the inference task consists in providing 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 photomultipliers that have been illuminated (hit) by the Cherenkov light cone emitted by a charged particle traversing the detector, as shown in Figure 39.

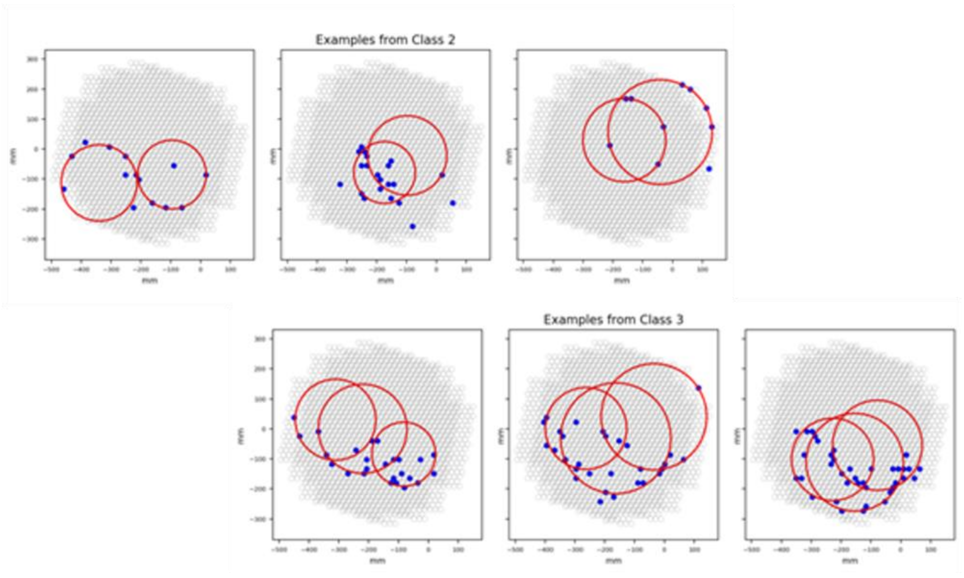


Figure 39 Examples of events belonging to class 2 and 3 (2 or >=3 charged particles) as detected by the array of RICH photomultipliers (blu dots are the hit photomultipliers, red circles are the tracks reconstructed offline by the NA62 experiment analysis software framework)

Figure 40 depicts the workflow for the generation of processing Kernels implementing neural networks designed for the inference tasks in RAIDER; these kernels are then integrated in the FPGA design as HLS kernels in the APEIRON framework, as described in deliverable D4.1.

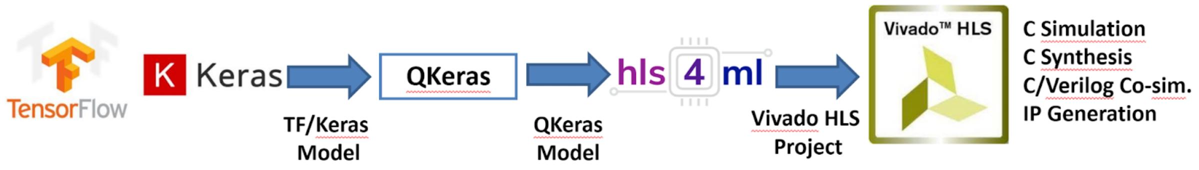


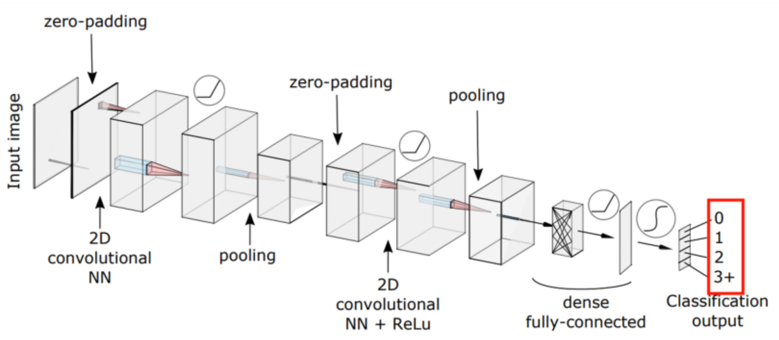
Figure 40 The workflow for the generation of NN kernels in RAIDER

Using this workflow, design targets (efficiency, purity, throughput, latency) and constraints (mainly FPGA resource usage) must be taken into account and verified at any stage:

1. TensorFlow/KERAS[12]: on this first stage the NN architecture (number and kind of layers) and representation of the input is designed, then using an appropriate training strategy (class balancing, batch sizes, optimizer choice, learning rate, etc.), the network

is trained and KPIs can be measured. If they don't meet the targets the process is repeated, modifying input representation and the NN architecture.

2. QKeras[13]: in this second stage, the original TF/Keras NN model is modified by searching iteratively the minimal representation size in bits of weights, biases and activations, possibly by layer that preserves the expected KPIs. For the RAIDER application, the neural network generated through this quantization step, yielded a neural network that uses an 8-bit fixed point <8, 1> representation for weights and biases and 16-bit fixed point <16, 6> for activations.
3. HLS4ML[14]: the QKeras model is translated into the corresponding Vivado HLS implementation (annotated C++ code). Several handles are available at this stage to guide the translation, e.g. tuning of REUSE FACTOR configuration parameter (low values yield low latency, high throughput, high resource usage design), also clock frequency can be set.
4. Vivado HLS[15]: C/Verilog co-simulation for rapid verification of performance and synthesis of kernel IP to be integrated in the APEIRON framework.



```
Layer (type)                 Output Shape              Param #
=================================================================
input1 (InputLayer)          [(None, 16, 16, 1)]       0

conv1 (Conv2D)               (None, 16, 16, 8)         80

act1 (Activation)            (None, 16, 16, 8)         0

maxp1 (MaxPooling2D)         (None, 8, 8, 8)           0

conv2 (Conv2D)               (None, 8, 8, 8)           584

act2 (Activation)            (None, 8, 8, 8)           0

maxp2 (MaxPooling2D)         (None, 4, 4, 8)           0

flatten (Flatten)            (None, 128)               0

fc3 (Dense)                  (None, 16)                2064

act3 (Activation)            (None, 16)                0

fc4 (Dense)                  (None, 4)                 68

softmax (Activation)         (None, 4)                 0

=================================================================
Total params: 2,796
```

Figure 41 Details of the designed Convolutional Neural Network model

Following this workflow, we designed a lightweight Convolutional Neural Network having just 2796 parameters and suitable to be implemented on a FPGA, along with the corresponding representation of the input data. The designed CNN model, represented in Figure 41, has been deployed on a Xilinx Alveo U200 FPGA with a very limited resource usage. This CNN receives as input a compressed representation of the original event in form of a B&W 16x16 image, as depicted in Figure 42.

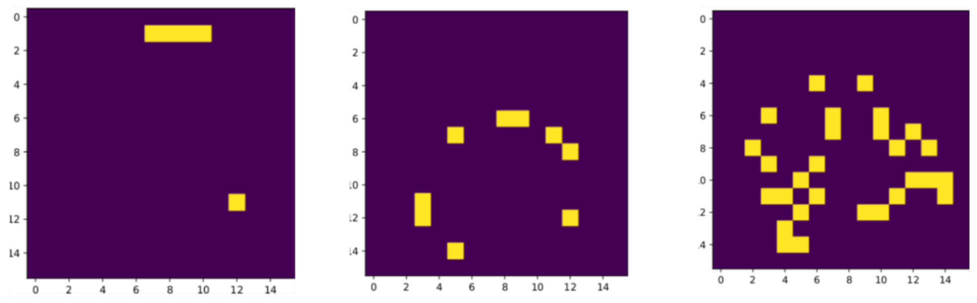


Figure 42 Example of input images for the CNN (left class 0, center class 1, right class 2).

3.6.1 Results

We report KPIs for three different configurations of processing devices (CPU, CPU+GPU and CPU+FPGA): KPIs evaluated on CPU and CPU+GPU represent the baseline for those evaluated from measurements on the CPU+FPGA reference configuration.
The application computational kernel is represented by the forward pass of the CNN, to infer the number of charged particles present in the input events. In detail, the performance of the three configuration of devices was measured by taking 2.7M events, extracted from those collected during past runs of the NA62 experiment, as neural network input and profiling the time to solution and the energy to solution to execute the inference task on the full dataset.
In the application testbench we setup to measure the application KPIs, shown in Figure 43, the host is in charge of moving events from its memory to the FPGA memory, then *Krnl_sender* HLS kernel on FPGA forwards them to the inference pipeline *Top_nnet*, finally inference results are stored in host memory by the *Krnl_receiver* HLS kernel.
The *Top_nnet* pipeline includes two stages: the first implementing the compressed encoding of events in 16x16 B&W images and the second executing the Convolutional Neural Network.
Interconnection between the sender and receiver kernels and the inference pipeline on FPGA is accomplished via the TEXTAROSSA communication IP (see deliverable D2.8 - *IP for low-latency internode communication links, part 1*).

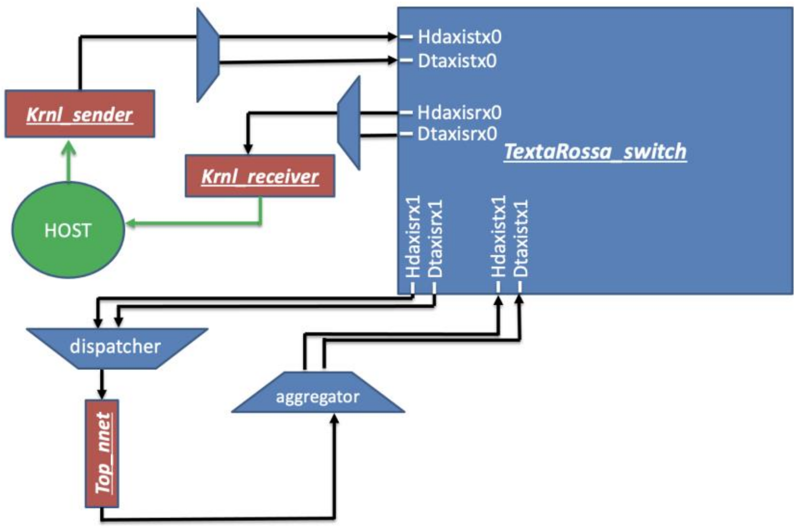


Figure 43 Testbench for the RAIDER The host is in charge of streaming events from its memory through *Krnl*_sender to the processing pipeline *Top_nnet* on FPGA and to collect results from *Krnl_receiver*.

3.6.1.1 *Power measurement methods*

To perform power measurements on each we used different methods, in detail:

1. CPU: LIKWID toolsuite (in particular the likwid-perfctr CLI), as described in deliverable D1.4 - *Power Measurement on CPUs (x86_64 Architectures: Running Average Power Limit (RAPL) interface).* The total power for the processor (Cores plus DRAM domain values) has been recorded.
2. GPU: GPowerU tool developed internally and based on the NVIDIA NVML library, available on github (https://github.com/crrossi/GPowerU).
   GPowerU is a simple tool able to measure the power consumption of a CUDA kernel in specific points of the device code and to generate the complete power profile, with timestamp values correlated with the start and end of the application's execution, differently from *nvidia-smi –query* functionality which runs "application blind", and with a finer time resolutions (tens of milliseconds vs. seconds). For additional information refer to deliverable D1.4 - *Power Measurement on GPU (NVML Library).*
3. FPGA: Xilinx Power Monitor Tools as described in D1.4 - *Power Measurement on FPGA (Xilinx Power Monitoring Tools).*

3.6.1.2 *Impact of using alternative hardware platforms on presented results*

In the application testbench we setup to measure the application KPIs, shown in Figure 43, the stream processing happens entirely in the FPGA and processing performance is not affected by the host system as long as it can sustain the PCIe Gen3 x16 bandwidth between its main memory and the FPGA accelerator. For these reasons we are confident that using our lab server instead the IDV-E system does not alter the measured KPI n.1 (throughput) significantly. Also, we do not expect appreciable differences in this regard using the Alveo U200 instead of the Alveo U280 FPGA card, since our application does not make use of the distinctive features of the latter (namely HBM memory).
For what concerns KPI n.2 (energy efficiency), the task demanded to the host in this version of the application testbench is to feed the FPGA design with data read from its main memory and to store the produced results. The host side will be set aside from the data stream handling in the final version of RAIDER, with data arriving from one FPGA network channel and results send to a consumer again through a network channel, and its tasks will be limited to platform management. Taking into account these considerations, we include the energy consumption of the host in reporting results for this version of the application, knowing that using our lab X86_64 server instead of the ARM based IDV-E yields a higher contribution to the total energy consumption, and reported KPI n.2 (energy efficiency) will be most likely slightly underestimated compared to the one reachable on IDV-E.

3.6.1.3 *Baseline KPIs*

Referring to the CPU only and CPU+GPU measurements, they were performed on the CNN Keras model invoking the Tensorflow *model.predict()* library call on the input data loaded on the CPU/GPU memory. By using the *tf.device()* function we were able to select the execution of the *model*.*predict()* function to be performed on CPU or GPU.
The execution time of the inference task on the full 2.7M events dataset was collected bracketing the *model.predict()* call with two calls to the python *time()* function and evaluating the difference between the second and first returned values, measured values are reported in Table 15 along with the corresponding throughput KPI.

The CPU and GPU power measurements, concurrent with the *model.predict()* execution, were performed respectively using the LIKWID and GPowerU profiling tools and then the integral of the power profile was used to compute the application energy-to-solution and energy efficiency. These power profiles are presented in Figure 44 for the CPU only configuration, Figure 45 and Figure 46 for CPU+GPU configuration, while the corresponding energy-to-solution and energy efficiency KPI values are reported in Table 15.
For the CPU+GPU configuration, the sum of the energy consumption of both devices is reported for the energy-to-solution.

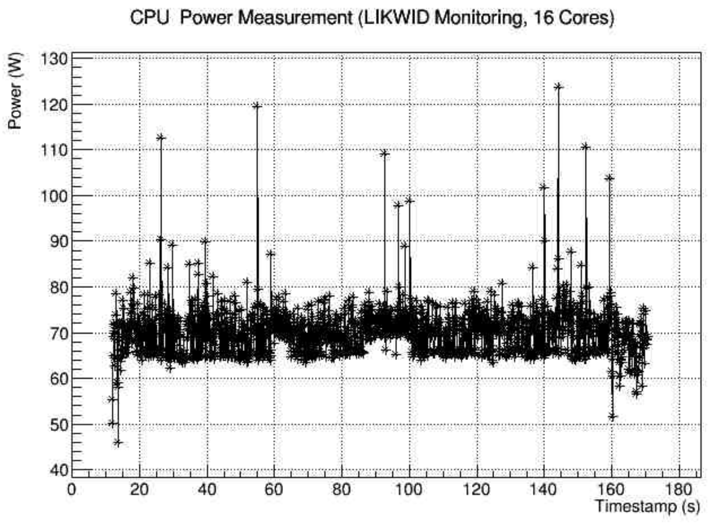


Figure 44 Power profiling of model.predict() function performed via LIKWID tool

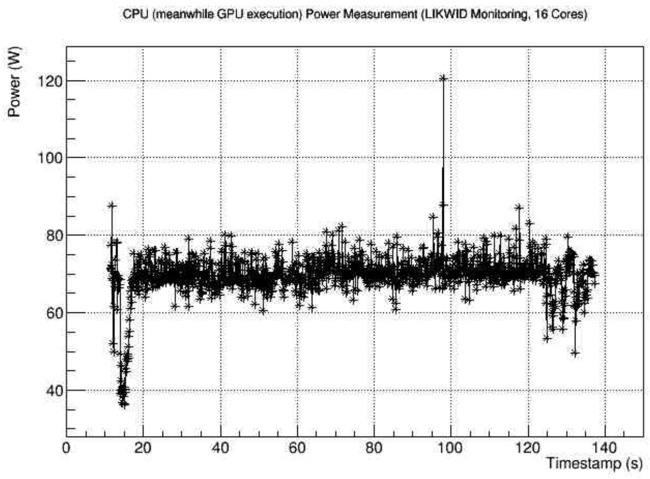


Figure 45 Power profiling of model.predict() function for the CPU

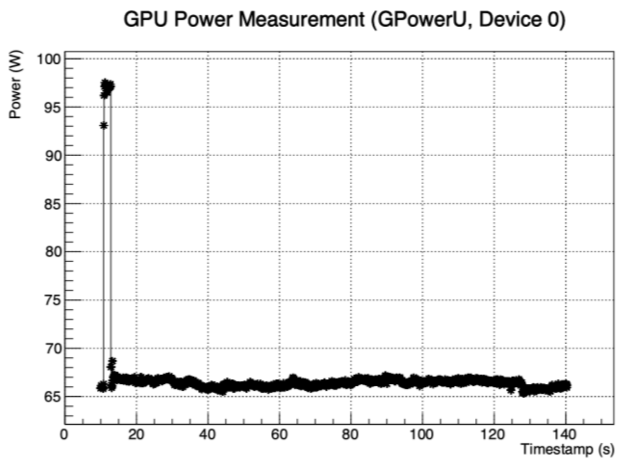


Figure 46 Power profiling of model.predict() function for the GPU

KPI	CNN CPU tensorflow	CNN CPU+GPU tensorflow
purity/efficiency (per class)	efficiency: - 0: 93% - 1: 83% - 2: 75% - 3+: 83% purity: - 0: 88% - 1: 90% - 2: 71% - 3+: 78%	efficiency: - 0: 93% - 1: 83% - 2: 75% - 3+: 83% purity: - 0: 88% - 1: 90% - 2: 71% - 3+: 78%
time to solution [s]	158.521	125.963
throughput [events/s]	**189250**	**238165**
energy to solution [J]	11091.919	17497.783 (8724.648 GPU)
energy efficiency [events/J]	**270.467**	**154.305**

Table 15 Baseline KPIs evaluated on the execution of the Keras model.predict() function **2.7M events for the CPU only and CPU+GPU configurations**

3.6.1.4 KPIs for the FPGA implementation

In the CPU+FPGA testbench configuration, the inference task is performed by the CNN deployed on FPGA through the workflow depicted in Figure 40 and using the same tensorflow/Keras model used to collect the two sets of baseline KPIs.
Besides the testbench configuration described above sporting a single *Top_nnet* inference processing pipeline (1CNN) and depicted in Figure 43, we considered the opportunity of deploying multiple instances of the pipeline given its limited usage of FPGA resources and the intra-device scalability offered natively by the APEIRON framework.

So we deployed and measured KPIs also for a configuration of the design that includes 2 inference processing pipelines (2CNN), as represented in Figure 47.

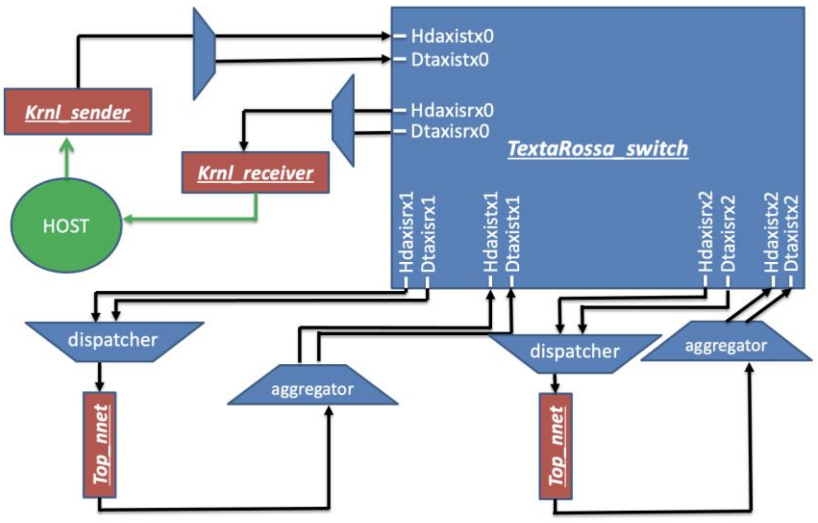


Figure 47 CNN RAIDER testbench integrating 2 inference processing pipelines

Percentage of the FPGA resources usage for the two design configurations are reported in Table 16 and Table 17.

Name	Kernel	LUT (9.57 % Used)	Register (4.25 % Used)	BRAM (6.57 % Used)	URAM (% Used)	DSP (10.23 % Used)
Textarossa_switch_1	Textarossa_switch	40640 (3.44%)	52273 (2.73%)	111 (5.79%)	0 (0.0 %)	0 (0.0 %)
aggregator_0_1	aggregator_0	226 (0.02 %)	839 (0.04 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
aggregator_1_1	aggregator_1	226 (0.02 %)	839 (0.04 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
dispatcher_0_1	dispatcher_0	314 (0.03 %)	945 (0.05 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
dispatcher_1_1	dispatcher_1	314 (0.03 %)	945 (0.05 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
krnl_receiver_1	krnl_receiver	1426 (0.12 %)	2348 (0.12 %)	2 (0.09 %)	0 (0.0 %)	0 (0.0 %)
krnl_sender_1	krnl_sender	1251 (0.11 %)	2051 (0.11 %)	2 (0.09 %)	0 (0.0 %)	0 (0.0 %)
top_nnet_1	top_nnet	71072 (6.01 %)	21183 (1.11 %)	13 (0.6 %)	0 (0.0 %)	700 (10.23 %)

Table 16 Percentage of FPGA resources used by the 1CNN RAIDER design configuration

Name	Kernel	LUT (15.8 % Used)	Register (5.53 % Used)	BRAM (7.17 % Used)	URAM (0 % Used)	DSP (20.46 % Used)
top_nnet_1	top_nnet	71072 (6.01 %)	21183 (1.11 %)	13 (0.6 %)	0 (0.0 %)	700 (10.23 %)
top_nnet_2	top_nnet	71072 (6.01 %)	21183 (1.11 %)	13 (0.6 %)	0 (0.0 %)	700 (10.23 %)
Textarossa_switch_1	Textarossa_switch_1	42621 (3.61%)	57587 (3.01%)	125 (5.79%)	0 (0.0 %)	0 (0.0 %)
aggregator_0_1	aggregator_0	226 (0.02 %)	839 (0.04 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
aggregator_1_1	aggregator_1	226 (0.02 %)	839 (0.04 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
aggregator_2_1	aggregator_2	226 (0.02 %)	839 (0.04 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
dispatcher_0_1	dispatcher_0	314 (0.03 %)	945 (0.05 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
dispatcher_1_1	dispatcher_1	314 (0.03 %)	945 (0.05 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
dispatcher_2_1	dispatcher_2	314 (0.03 %)	945 (0.05 %)	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)
krnl_receiver_1	krnl_receiver	1426 (0.12 %)	2348 (0.12 %)	2 (0.09 %)	0 (0.0 %)	0 (0.0 %)
krnl_sender_1	krnl_sender	1424 (0.12 %)	2449 (0.13 %)	2 (0.09 %)	0 (0.0 %)	0 (0.0 %)

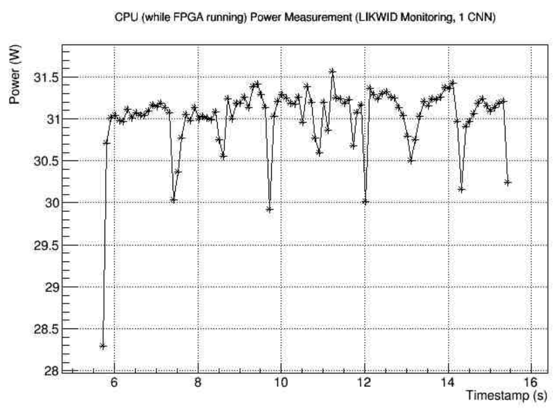
Table 17 Percentage of FPGA resources used by the 2CNN RAIDER design configuration

In both configurations, input data are loaded on the FPGA memory from the CPU HOST memory via XRT functions then, they are sent through the intra-FPGA network via an HLS kernel (*krnl_sender*) which, in multiple CNN case, takes care of the load balancing by sending data Round-Robin to the multiple kernel replicas. Lastly, after the inference processing, another HLS kernel (*krnl_receiver*) receives data coming from the network and stores them back to the

CPU HOST memory, where they are collected in order to evaluate efficiency and purity results for each of the labeled classes.
Both setups share the same clock frequency of 100 MHz.

The execution time has been measured on the host using the *std::chrono::high_resolution_clock::now()* method, with the start time corresponding to the launch of the *krnl_sender* (immediately after the completion of events data loading on FPGA memory) and with the end time at the completion of the *krnl_receiver* writing of inference data back to the host memory.
The throughput and time-to-solution values for the two configurations, which are reported in Table 18.

The FPGA setups power measurements, concurrent with the CNN kernel(s) execution, were extrapolated from the XRT summary .csv output file in order to produce the power profiles depicted in Figure 48 and Figure 49 respectively for the 1CCN and 2CCN design configurations.
The integrals of the power profiles have been used to compute the energy-to-solution and energy efficiency values for both setups, which are reported in Table 18.



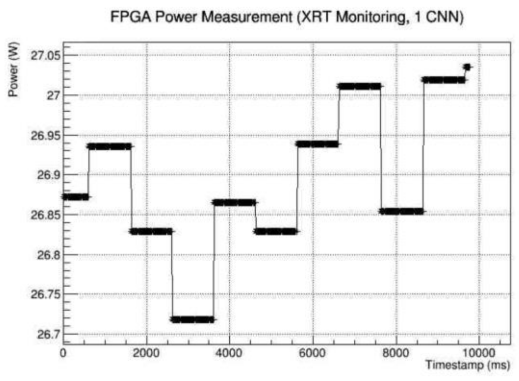
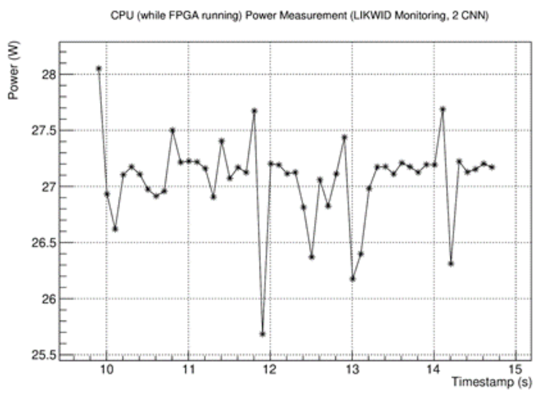


Figure 48 Power profiling of 1CNN RAIDER design configuration CPU (left) and FPGA (right)



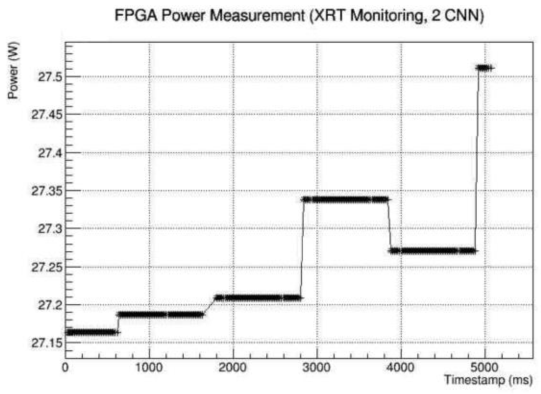


Figure 49 Power profiling of 2CNN RAIDER design CPU (left) and FPGA (right)

KPI	CPU+FPGA (1CNN) RAIDER	CPU+FPGA (2CNN) RAIDER
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**	**10079.328**

Table 18 KPIs for the two design configurations of RAIDER

Finally, in Table 19, we consider the improvement of the measured KPIs on the two design configurations (1CCN, 2CCN) over the two sets of baseline KPIs (CPU only and CPU+GPU).

Design/KPI	**Improvement factor over CPU**	**Improvement factor over CPU + GPU**
1CNN/ throughput	1.473	1.169
1CNN/ energy efficiency	17.726	31.070
2CNN/ throughput	2.913	2.135
2CNN/ energy efficiency	37.267	65.332

Table 19 Improvement factors of the KPI for the 1CNN and 2CNN designs over the baseline

3.7 TNM – INFN

The Quantum TEA is a suite of applications that utilizes tensor network methods (TNM) to simulate quantum systems and solve machine learning problems. Among these applications is the Quantum Matcha TEA, which is an emulator for quantum computers that is powered by matrix product states. To assess the performance of our application, we tested it with a Quantum Fourier Transform (QFT) algorithm on a set of entangled qubit blocks, with each block consisting of N entangled qubits. The total number of qubits for the QFT algorithm consists always of 100 qubits to evaluate its effectiveness on non-trivial states.

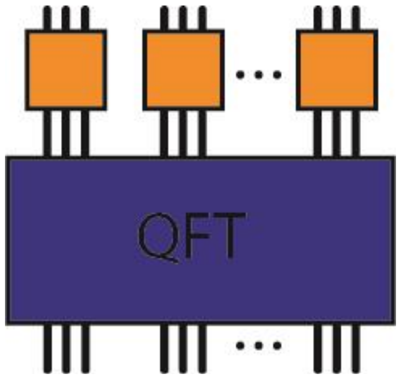


Figure 50 Quantum Fourier Transformation (QFT) algorithm acting on a block of N entangled qubits. Here in the example, we show N=3 where each qubit is represented by one vertical line and the entangling blocks are displayed in orange.

Here in the Figure 50, we show N=3 where each qubit is represented by one vertical line and the entangling blocks are displayed in orange. The timeline from the perspective of each qubit, represented by a vertical black line, goes from the top downwards: first, the qubit is entangled with (N-1) neighbors via an entangling block and then the QFT is executed on all 100 qubits.. Apart from the boundary effects, we have around 100 / N entangling blocks.

We present results obtained using the Dibona configuration's computing node. The performance of the system is measured by using LIKWID, which determines energy consumption as an indicator of efficiency. To measure the effectiveness of our results, we have chosen the number of gates per second (Gates/s) and the number of gates per Watts (Gate/Ws) as the key performance indicators (KPIs). This metric allows us to evaluate the efficiency of our execution when dealing with entangling blocks of different sizes N. In particular, we have conducted tests on circuits with N=6 and N=8.

The setup '*likwid-perfctr -f -C 0 -g ENERGY -t 100ms -O -o data.likwid ./qmatchatea.exe*' executes the simulation on one core with a measurement frequency of 100ms. The quantum matcha tea executable is compiled with gfortran v8.4 and quantum match tea itself is based on version v0.3.7.

In Figure 51 and Figure 52, we have included plots that display the energy consumption data we have collected for single-CPU baseline. These plots offer a visual depiction of our system's energy consumption over time for N=6 and N=8, enabling us to observe that the energy usage remains relatively constant, except for minor fluctuations in both cases.

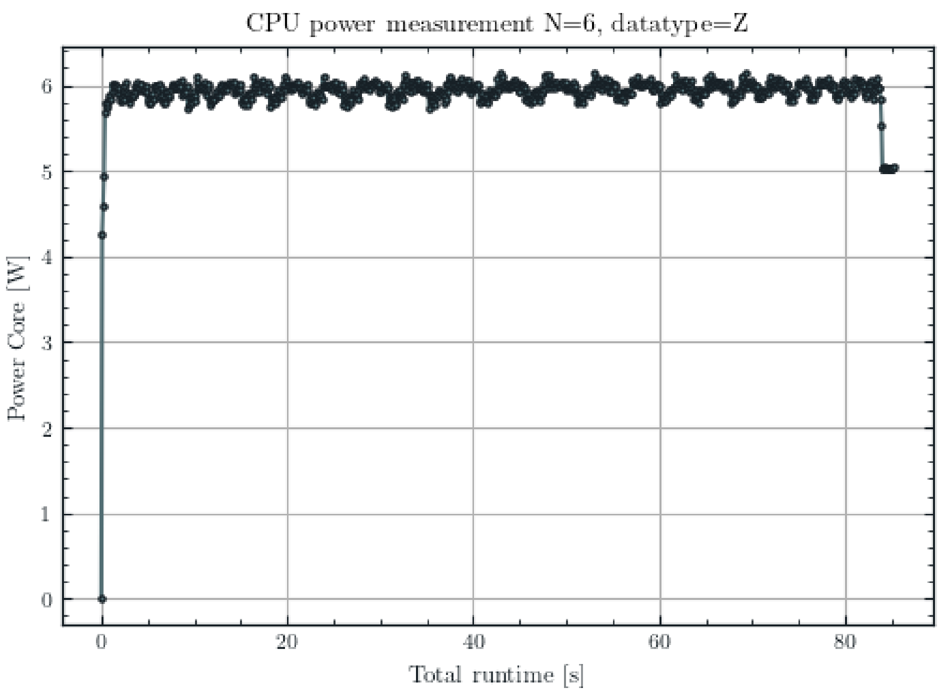


Figure 51 Power measurement for a single-core double-precision simulation of a quantum Fourier transformation of 100 qubits on blocks of six entangled qubits.

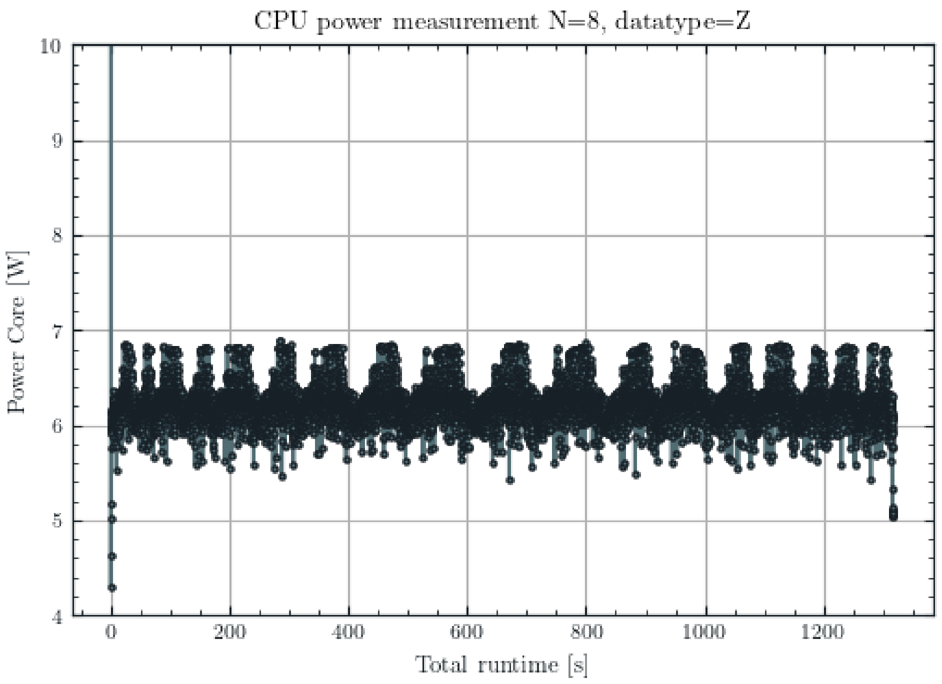


Figure 52 Power measurement for a single-core double-precision simulation of a quantum Fourier transformation of 100 qubits on blocks of eight entangled qubits.

We have provided a summary of the results in the Table 20. This table outlines the key metrics we have collected and provides an overview of the performance of our system. In addition to the energy measurements, we have executed the single-CPU baseline without power measurement to obtain an unbiased computation time.

Description	N = 6, data type = Z	N = 8, data type = Z
Number of gates	11850	10995
Time (mm:ss)	1:25	21:52
Gates/s (KPI from D6.1)	139.4	9.0
Power consumption Ws	504.8	8200.5
Gates/Ws (KPI from D6.1)	23.5	1.3

Table 20 Properties for single-core double-precision simulations of a quantum Fourier transformation of 100 qubits on blocks of N entangled qubits. The number of gates executed during the algorithm together with the execution time and power consumption leads to two KPIs defined for tensor network simulations.

We see the difference between the results with different N both in execution time and in power consumption. As expected from a quantum physics point of view, the computation time, and thus energy consumption, depends drastically on the amount of entanglement present in the system, where the entanglement increases with even numbers of N. For future benchmark, it is important to compare the same protocol, i.e., the number of entangling blocks controlling the entanglement and the QFT algorithm running thereafter..

The results shown so far have been for double precision simulation where we truncate singular values below 10^{-9} in the compression scheme, which is a key part of our algorithm.

As our next step of analysis, we compare the performance of single precision (C) and double precision (Z) data types. We adapt the truncation threshold to 10^{-5} for both data types which guarantees to truncate numerical noise also for single precision simulations. The Figure **53**, Figure 54 and Table 21 compare the data types.

Description	N=6, C	N=6, Z	N=8, C	N=8, Z
Number gates	11850	11850	10995	10995
Time (mm:ss)	0:30	0:32	7:36	7:51
Gates/s	395.0	370.3	24.1	23.3
Power Ws	170.7	183.5	2665.4	2883.1
Gates/Ws	69.4	64.6	4.1	3.8

Table 21 Comparison of single-precision (C) versus double-precision (Z) simulations on a single core for a quantum Fourier transformation on blocks of N entangled qubits. The comparison shows minor benefits for using single precision in terms of computation time and energy consumption.

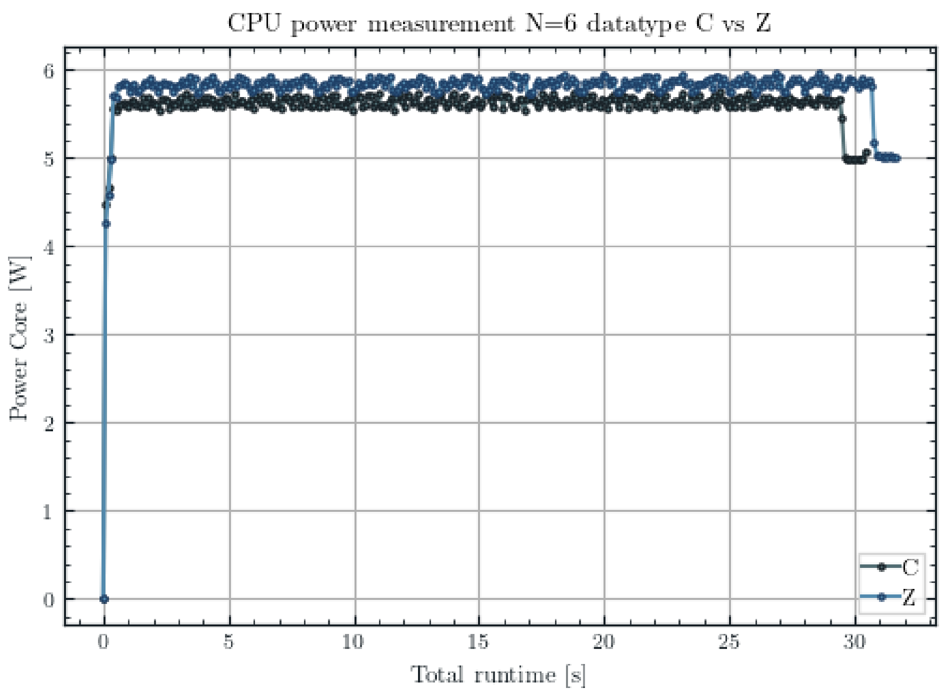


Figure 53 Power measurement for a single-precision (C) and double-precision (Z) simulation on a single core of a quantum Fourier transformation of 100 qubits on blocks of six entangled qubits.

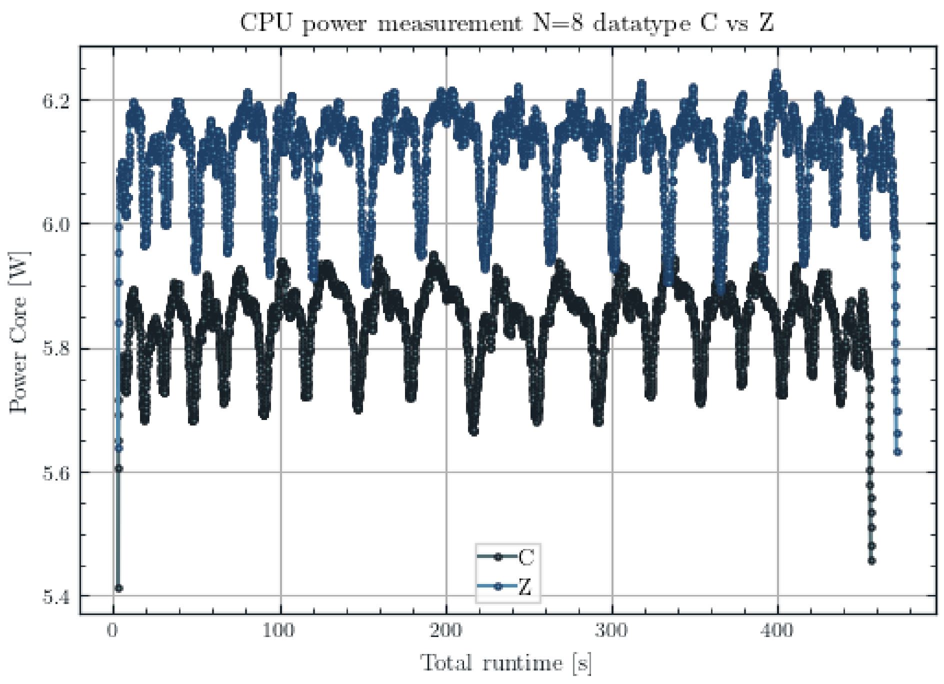


Figure 54 Power measurement for a single-precision (C) and double-precision (Z) simulation on a single core of a quantum Fourier transformation of 100 qubits on blocks of eight entangled qubits.

The comparison between the single and double precision shows a slight advantage of the single precision in terms of runtime and energy consumption. Together, they add up to around 7% fewer energy consumption for single precision simulations. For the plot of the power measurement for N=8, we apply a moving average to suppress fluctuations in the energy consumption and distinguish single precision versus double precision data. A possible next step is to compare the convergence of both simulations with respect to their results.

This benchmarking process allowed us to evaluate the effectiveness of our emulator under different conditions and identify areas where further improvements could be made. Additional KPIs defined in D6.1 as Qbits/s or Qbits/Ws are not shown here. These KPIs are more applicable to condensed matter problems where one wants to study the properties of the system while scaling in the number of qubits. The benchmarks of these simulations for condensed matter problems are not yet available. The energy measurements and benchmarks with GPUs are also not available yet.

3.8 ScalFMM (Mathlibs-INRIA)

ScalFMM is an HPC application developed at Inria that implements the fast multipole method (FMM), which enables computing pairwise interactions between N particles with quasi-linear complexity. Before the current project, ScalFMM was already parallelized with StarPU and had GPU capability.

Like Chameleon, we are using ScalFMM to validate our scheduler multreeprio, and we plan to extend it to study the exploitation of FPGA with StarPU.

We provide results obtained on computing nodes of older generations compared to Dibona (2 × 16-core Intel Skylake + 2 × V100 and 2 × 12-core Intel Haswell + 4 × K40). Our work here does not focus on the efficient implementation of computational kernels (i.e. optimizing for a given processing unit), but rather on the flexibility and robustness of the multreeprio scheduler. What matters in a configuration is not just the raw performance of the processing units, but also their number and the performance difference between the different types.

Since this is still ongoing work, we show results for two variants of the scheduler called lamtp{1,2} in Figure 55 that shows the number of particle interactions per second for different StarPU schedulers. As the simulation relies on the FMM, many of the interactions are computed approximately. Our scheduler is competitive with DMDA (a modified version of the Heft scheduler [16]), which is considered the most efficient StarPU scheduler, as well as the (la)heteroprio scheduler, a scheduler that was originally designed for the FMM [17,18].
We want to remind you that configuring the heteroprio scheduler manually is required, which is not necessary for laheteroprio or multreeprio, making things easier for developers.

On the V100 configuration, we obtain an efficiency of 230.10^9 interactions/Watt using the two GPUs and the two CPUs with multreeprio. We currently do not have a comparison with the other schedulers, but the related paper will include it.

Our next steps will be: 1) to port some FMM kernels to FPGA and 2) to adapt the multreeprio scheduler to consider energy in its decision-making process.

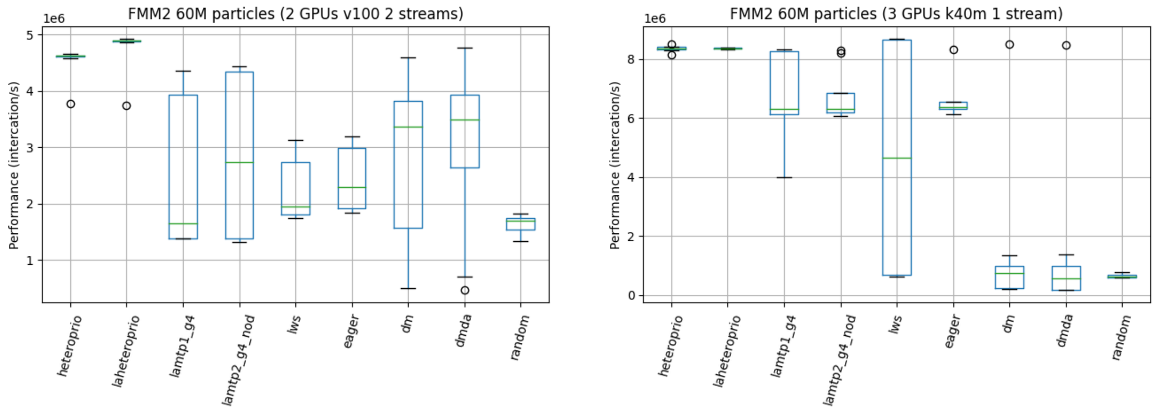


Figure 55 Interactions/s of a n-body simulation using ScalfMM and StarPU on two different hardware configurations. We compare our new scheduler (lamtp) against its legacy version (heteroprio) and the most used scheduler (DMDA).

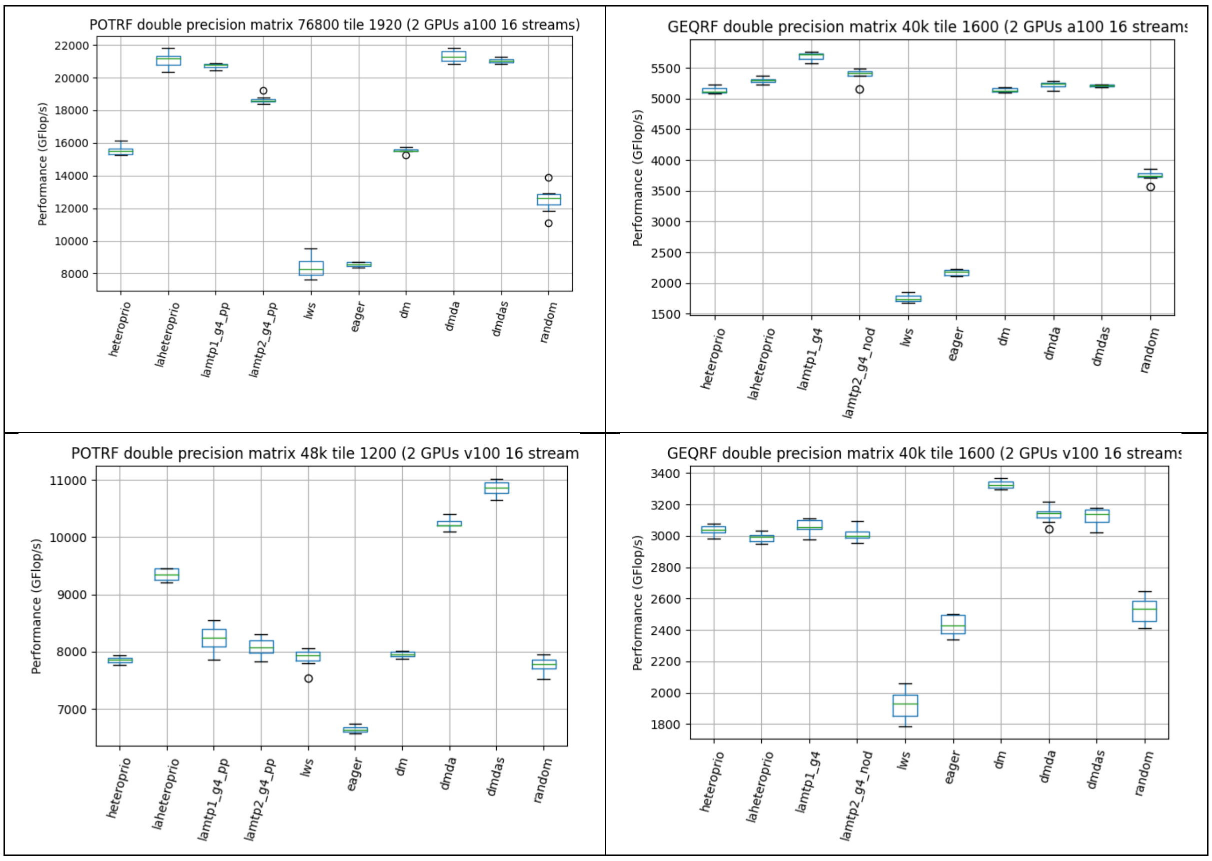
3.9 Chameleon (Mathlibs-INRIA)

Chameleon is a C library that provides parallel algorithms to perform BLAS/LAPACK operations, fully exploiting modern architectures. It had already been parallelized with StarPU before the start of the current project.

Similar to ScalFMM, we use Chameleon to validate our scheduler and plan to test it on FPGA and energy optimization. In Figure 56, we provide the Flops/s, for the Cholesky factorization (POTRF) and the QR factorization (GEQRF) on three different hardware configurations and different schedulers. DMDA is the most effective scheduler known for executing Chameleon. Our results show that our scheduler multreeprio (lamtp) is competitive with DMDA when the performance difference between CPU and GPU is low (K40), but is not as efficient as DMDA when the GPU is much more efficient than the CPU (V100).

On the V100 configuration, we obtain an efficiency of 21GFlops/Watt using the two GPUs and the two CPUs using multreeprio.

Similar as with ScalFMM, our next steps will be: 1) to port some GEMM kernels to FPGA and 2) to adapt the multreeprio scheduler to consider energy in its decision-making process. Additionally, we are currently investigating ways to improve execution on this type of configuration by offloading more tasks to the GPU.



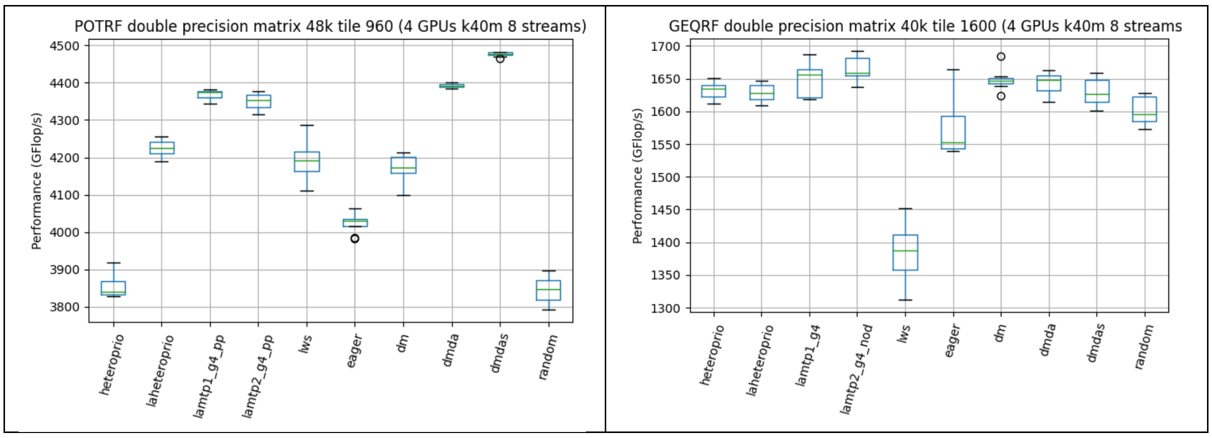


Figure 56 Flops/s for the Cholesky and Qr factorization (Chameleon) and StarPU on two different hardware configurations. We compare our new scheduler (lamtp) against its legacy version (heteroprio) and the most used scheduler (DMDA).

3.10 UrbanAir – PSNC

The UrbanAir application is tailored towards assessing and predicting air quality over complex urban areas. It is a multiscale model which benefits from coupling between WRF and EULAG. WRF is a community mesoscale weather prediction model, and its purpose in UrbanAir is to feed the latter model with meteorological fields. EULAG, the all-scale geophysical flow solver, is used to precisely model the wind flow over complex buildings shapes using immersed boundary method, and to calculate transport of contaminants. In the scope of the project we are focusing on the EULAG and exploiting heterogeneous resources in particular.
The dynamical core of EULAG constists of the MPDATA method and GCR solver. The latter consists of five routines: *prforc*, *divrhs*, *laplc*, *precon* and *reduction:*
Prforc – used to initialize the solver, applies boundary conditions and guesses the first updated velocity;
Divrhs – initializes the solver and computes the initial residual error of the elliptic problem;
Laplc – iteratively evaluates the generalized Laplacian operator;
Precon – accelerates the convergence of the variational scheme.

The original code written in Fortran 77 was previously rewritten to C++ in order to be able to run on different hardware architectures and proof-of-concept adaptation to heterogeneous resources was made [19]. The original code uses MPI parallelization. In order to support heterogeneous resources, including GPUs and many-cores, a stencil framework was introduced being responsible for parallelization and communication. Each kernel can be provided in either CPU or GPU realization (or both), and it is up to the framework to orchestrate proper execution. The kernels were adapted to GPUs by the means of CUDA. To exploit intra-node parallelization, OpenMP was used. Thus, within a node, CPU code is parallelized using OpenMP, and for GPU code using CUDA. The communication between GPUs and nodes is done via MPI. The computational domain is divided between all given hardware resources, where each process receives it’s part of the global domain called subdomain. Important to say, we decided to keep the static decomposition of the domain, i.e. partitioning is done before compilation process, as it allows the compiler to optimize the stencil loops.

Within Textarossa, a new testcase has been introduced which corresponds to the air quality use case. In contrast to previous studies, the computational domain is latter flat, as the domain increases horizontally in each direction, but not vertically. The testcase is about modelling air quality in urban environments, therefore 64 grid points (corresponding to domain height) is enough to model flows at buildings height. Improvements in communication framework were required to allow for efficient data exchange between multiple accelerators available within a single node and between multiple nodes. To efficiently exploit multi-core and multi-GPU within a node, automating mapping for cores and GPUs were provided for the most efficient realization. Finally, improvements in GPU kernels were made to improve correctness of results and their efficiency.

The baseline for benchmarks is a CPU version running on single and multiple nodes, combining MPI and OpenMP for inter- and intra-node communication respectively. Tests were conducted on currently available architectures, including testbed developed in Textarossa project. We used Altair, a PSNC’s cluster equipped with Intel Intel Xeon Platinium 8268 processors (2x24 CPU cores) and NVIDIA V100 cards. The Textarossa testbed is Dibona machine equipped with AMD processors and NVIDA GPU cards, see Section 2 for details. As described in D6.1 deliverable, our interest is to measure and improve in terms of GCRK number of iterations per second and per Watt. With the developed new architectures, we do not expect only improvement in computational performance and energy savings, but also a possibility to run larger domains on accelerators.

The tests are divided into strong scalability, weak scalability and energy consumption. If not explicitly stated, each test runs 100 iterations over 59M grid points domain (for strong scalability) or 1M grid points domain (for weak scalability). It corresponds to 960x960x64 and 128x128x64 domains respectively. For the energy measurements tests were additionally conducted using 500 iterations and over 15M grid points domain (for weak scalability).

3.10.1 Strong scalability

The parallelization on a single CPU node is done via OpenMP. Before running tests in multinode environment, the most efficient number of OpenMP threads needs to be chosen. In Figure 57, strong scalability within a single node is presented. The top compares execution time for different number of OpenMP cores used, while the bottom depicts achieved speedup. Increasing the number of threads lowers the execution time, although the more threads are used the less speedup is achieved. The optimal number of threads for a single node is 32 threads, although Altair and Dibona are equipped with 48 physical cores.

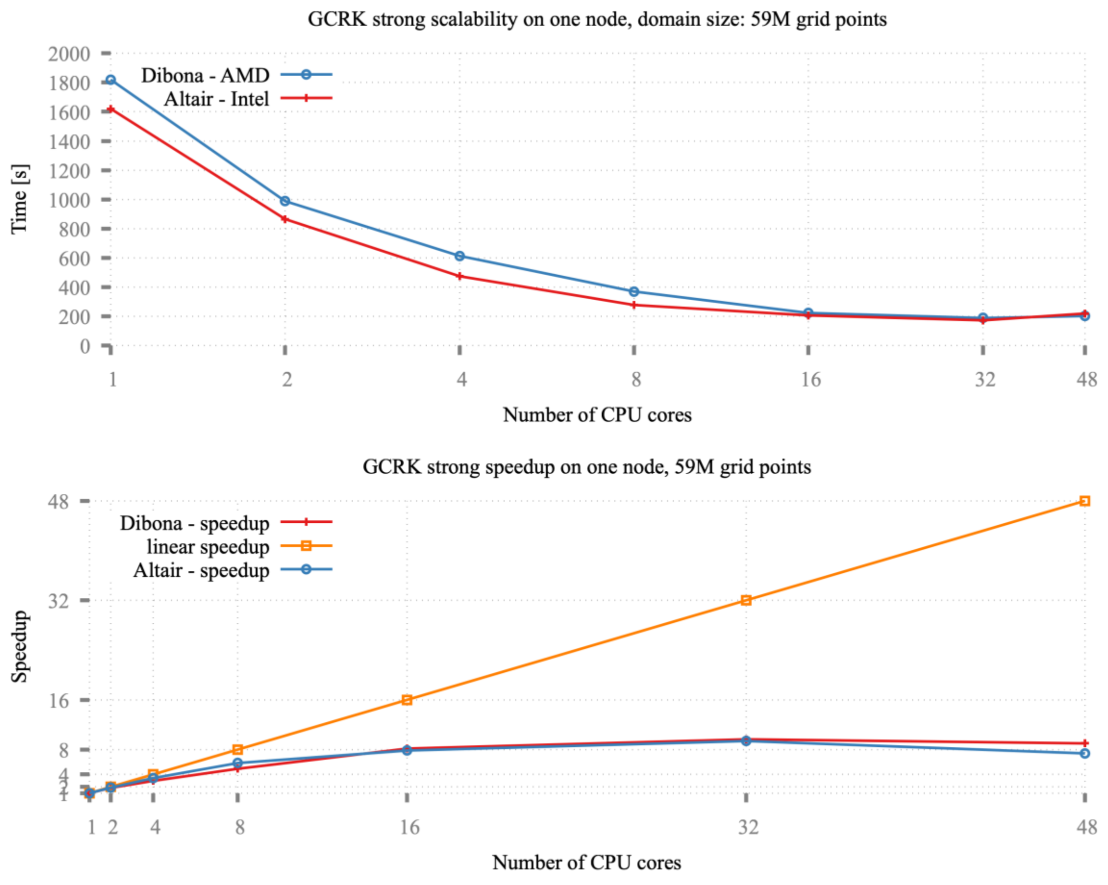


Figure 57 GCRK strong scalability on a single node. Top: execution time. Bottom: speedup.

The strong scalability study in multinode environment is conducted on Altair system up to 64 nodes. Within a node, code is parallelized using OpenMP paradigm with number of threads set to 32 CPU cores. The computational job is divided between the nodes, where each node receives its own and equal subdomain to calculate over, and the data is exchanged via MPI. The strong scalability results are presented in Figure 58. The top chart depicts execution time, while the bottom achieved speedup. Adding more nodes results in shorter execution time, but again the more nodes are used the less speedup is achieved. However, required time for obtaining results is getting shorter.

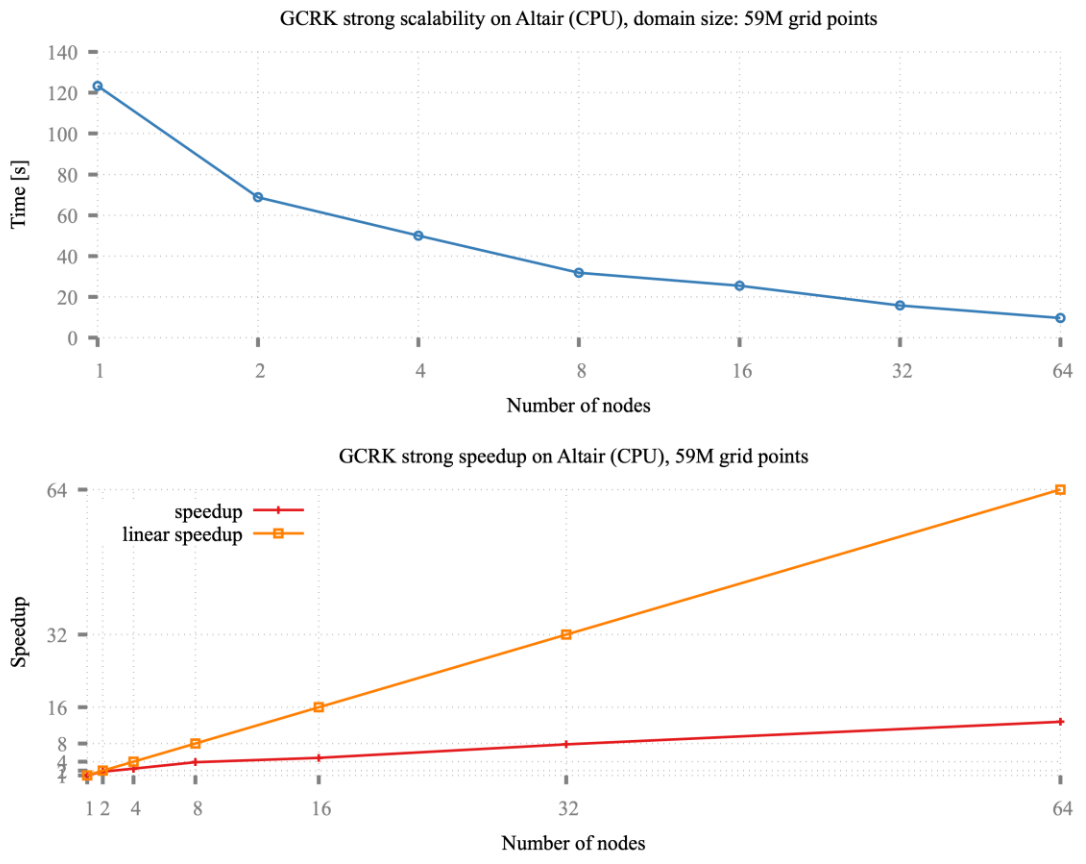


Figure 58 GCRK strong scalability on multiple nodes. Top: execution time. Bottom: speedup.

The strong scalability tests for the GPUs were performed on Altair and Dibona systems. It is important to say that the parameters for tests are different for each testbed. On Altair, a single GPU V100 is able to handle 15M grid points domain, while Dibona GPU computes over 59M grid points domain. Nonetheless, we would like to study whether the speedup characteristic is similar on different GPU architectures.

In Figure 59, strong scalability of the GCRK running up to 4 A100 GPUs is presented, which the achieved speedup closed to a linear one. In Figure 60, results for strong scalability up to 8 NVIDIA V100 GPUs is presented. On both figures the top chart presents how execution time changes when adding more GPU cards, while the bottom one depicts achieved speedup. On the older GPU architecture, the speedup is not as exceptional as on a modern one.

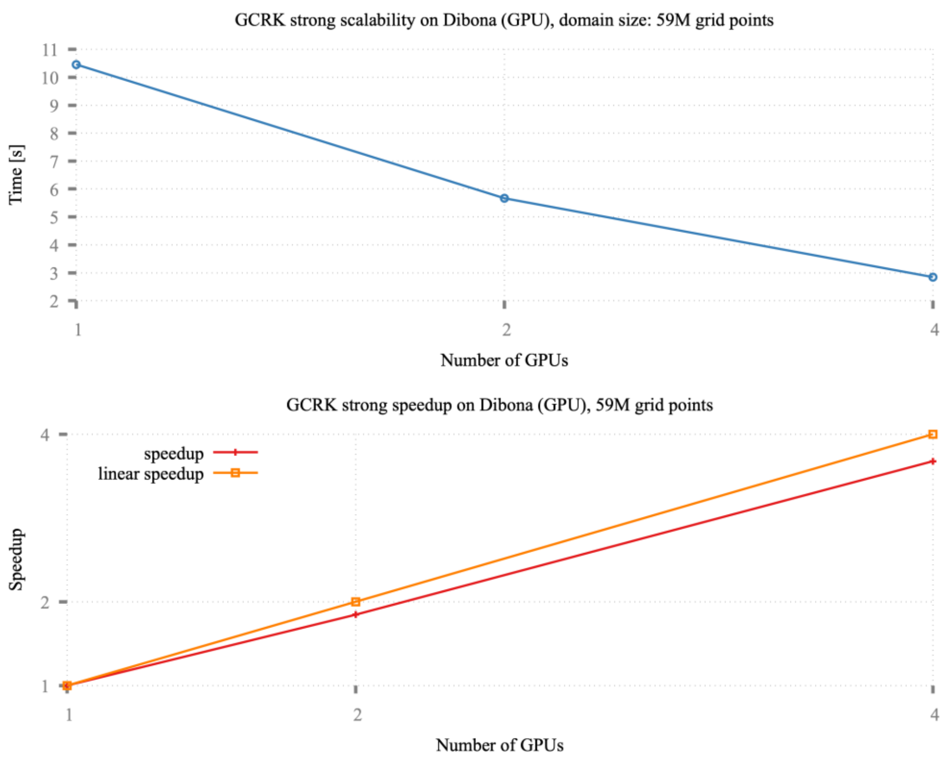


Figure 59 GCRK strong scalability on multiple A100 GPUs. Top: execution time. Bottom: speedup.

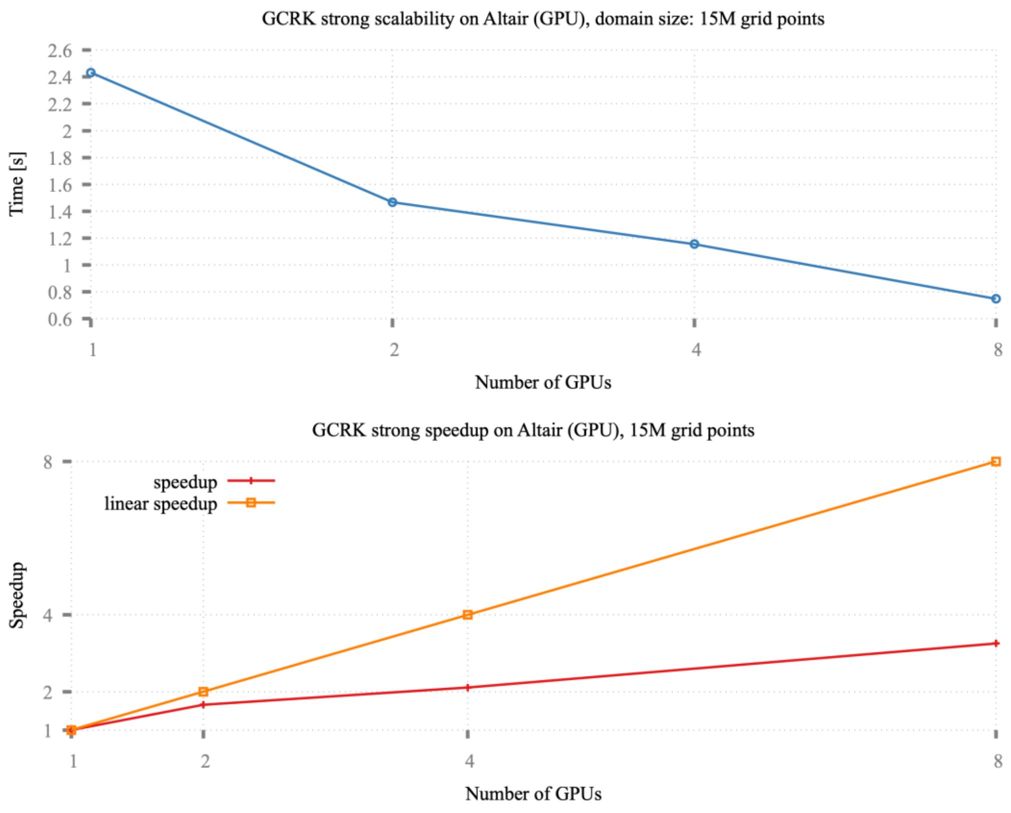


Figure 60 GCRK strong scalability on multiple NVIDIA V100 GPUs. Top: execution time. Bottom: speedup.

Errore. L'origine riferimento non è stata trovata.

3.10.2 Weak scalability

To compare weak scalability across different available hardware, 1M grid points problem per node or per GPU was selected. Figure 61 and Figure 62 presents results for CPUs and GPUs respectively. The top chart presents how execution time changes when more CPUs and GPUs are used, while the bottom depicts efficiency. In both cases, the more CPU nodes or GPUs are used, the weak efficiency drops more. Interestingly for Dibona GPU, while strong scale speedup is close to linear, the weak efficiency is the worst one among tested GPUs. The reason for this is the fact the chosen number of grid points (1M per node or GPU) is far from this GPU capabilities. This is why we conducted another weak scalability tests, aiming at solving 59M grid points per node or GPU. The results are presented in Figure 63, and one can observe that fitting GPU accelerator with more work (larger job to do) results in better efficiency.

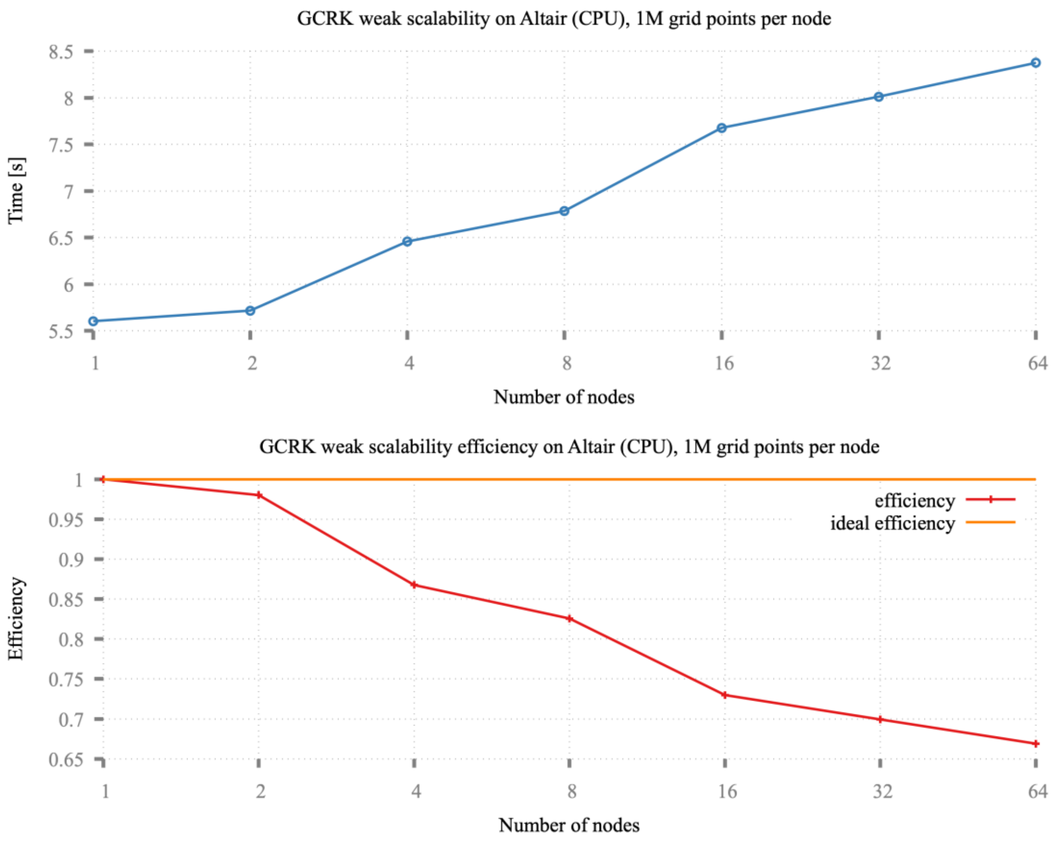
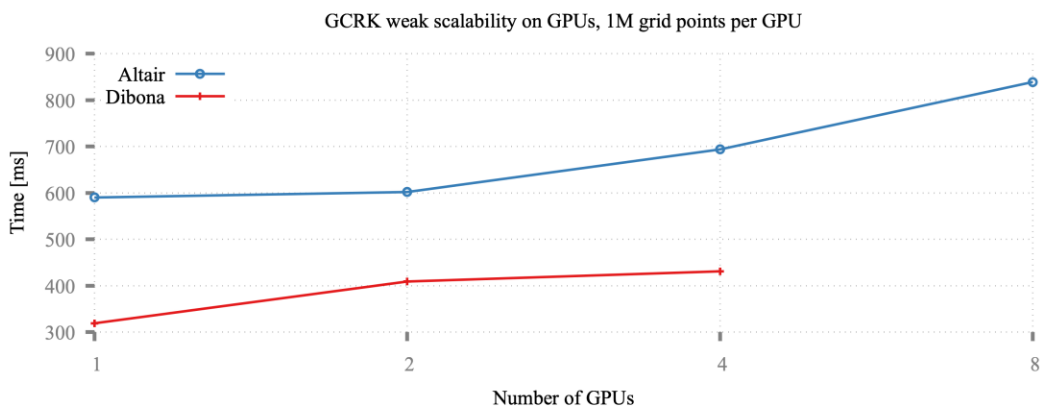


Figure 61 GCRK weak scalability on CPUs. Top: execution time. Bottom: efficiency.



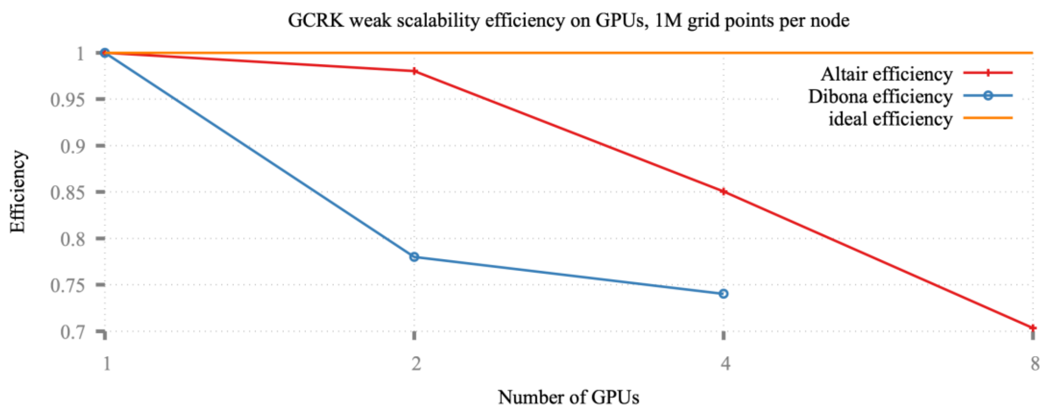


Figure 62 GCRK weak scalability on GPUs. Top: execution time. Bottom: efficiency.

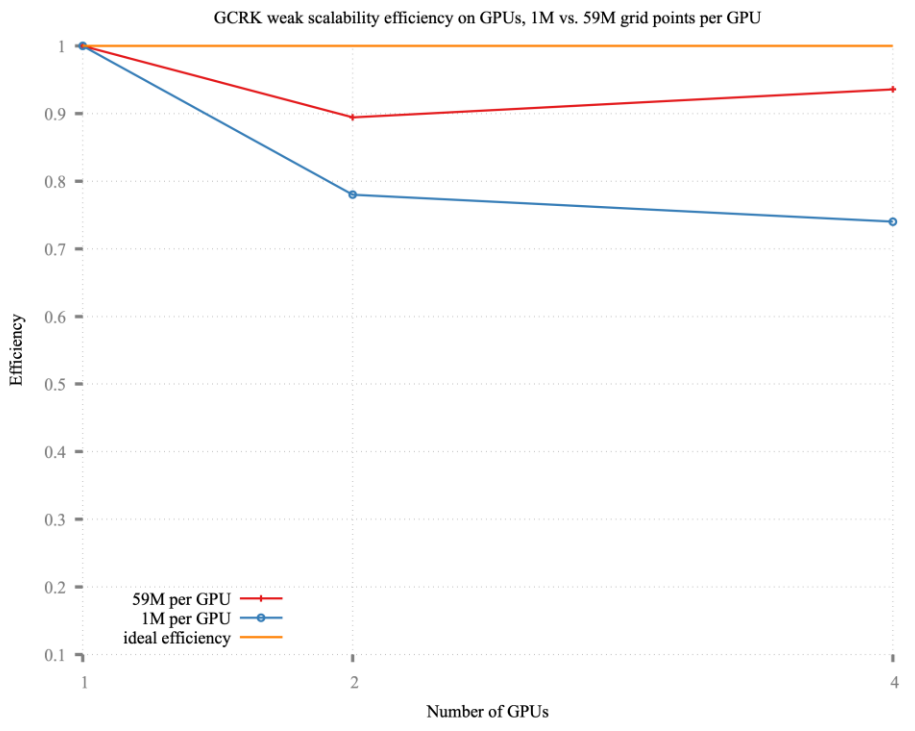


Figure 63 GCRK weak scalability on GPUs – comparison between 1M and 59M grid points problem

3.10.3 Energy measurements

Energy measurements were conducted on Dibona system using methodology described in D1.4. These experiments are to analyze energy and power consumption to find the sweet spot between computational performance and energy efficiency.

Figure 64 and Figure 65 present power consumption characteristic for the fixed amount of grid points per GPU, 1M and 15M points respectively. Each subfigure depicts what is the power consumption during GCRK execution when 1, 2 or 4 GPUs cards are used. For individual accelerators, the power consumption remains at the same level when more than one GPU is used, which means they are occupied with work at the same level. The power consumption is larger when a single GPU is used.

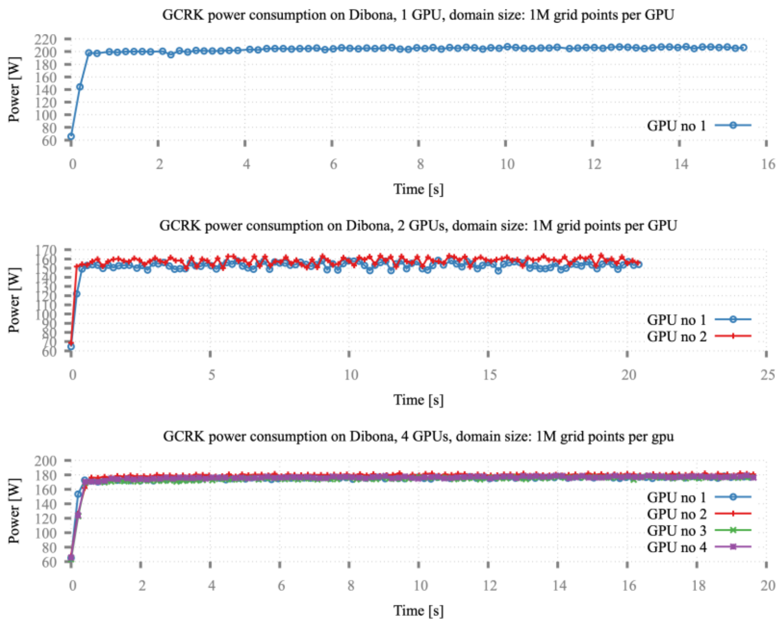


Figure 64 GCRK power consumption on GPUs, 1M grid points per GPU. Top: 1GPU, middle: 2 GPUs, bottom: 4GPUs

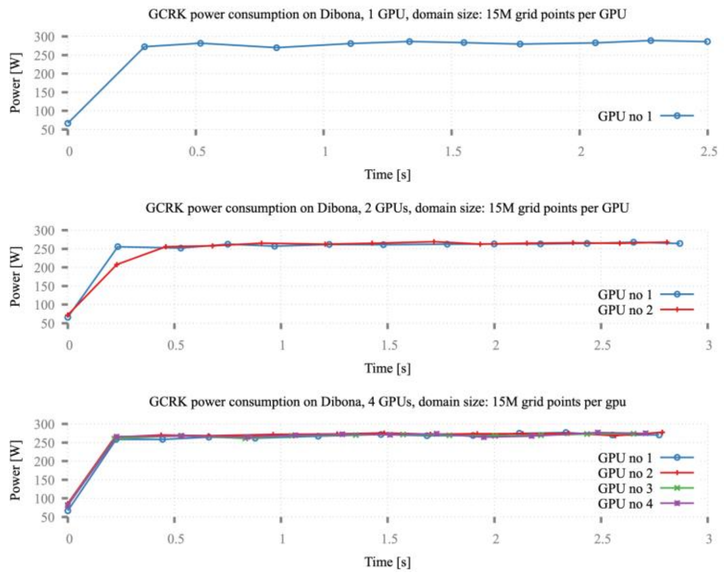


Figure 65 GCRK power consumption on GPUs, 15M grid points per GPU. Top: 1GPU, middle: 2 GPUs, bottom: 4GPUs

The same test was applied to a fixed problem size of 59M gridpoints, which is then equally divided between available GPUs. Results presented in Figure 66 demonstrates that power consumption characteristic is the same, no matter a strong or weak scalability problem is solved. Each subfigure presents power consumption during the execution for each GPU accelerator used.

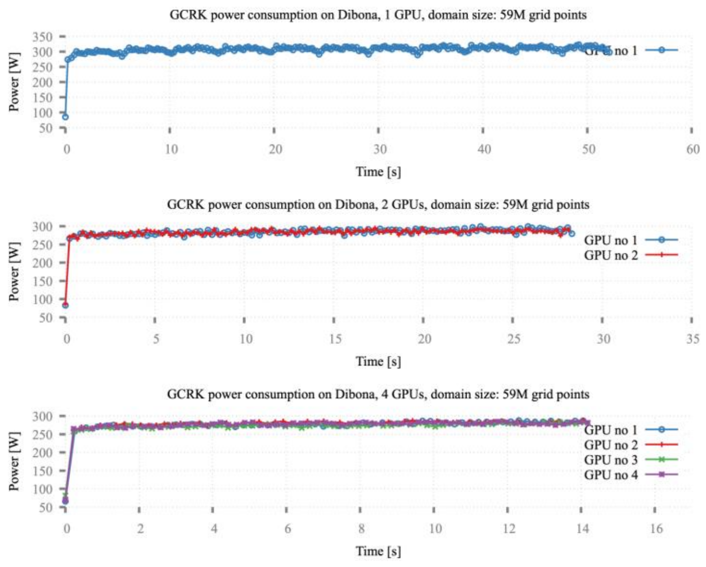


Figure 66 GCRK power consumption on GPUs, 59M grid points per job. Top: 1GPU, middle: 2 GPUs, bottom: 4GPUs

Next, we provide a study on what is the CPU energy power consumption when running kernels on GPU. Figure 67 presents such comparison for a 59M grid points size problem run on GPUs (up to 4) and OpenMP version of GCRK (up to 32 physical CPU cores). The power usage for CPU cores for OpenMP and GPU version is alike, and it is worth doing more tests in the future to check whether further increase in number of GPUs will results in similar increase in CPU power consumption. The energy accounted to package is constant for GPU realization of the GCRK. For the CPU implementation, package power consumption is significantly lower when less CPU cores are used, but when the optimal number of cores is used (32), package power consumption is even greater comparing to GPU realization.

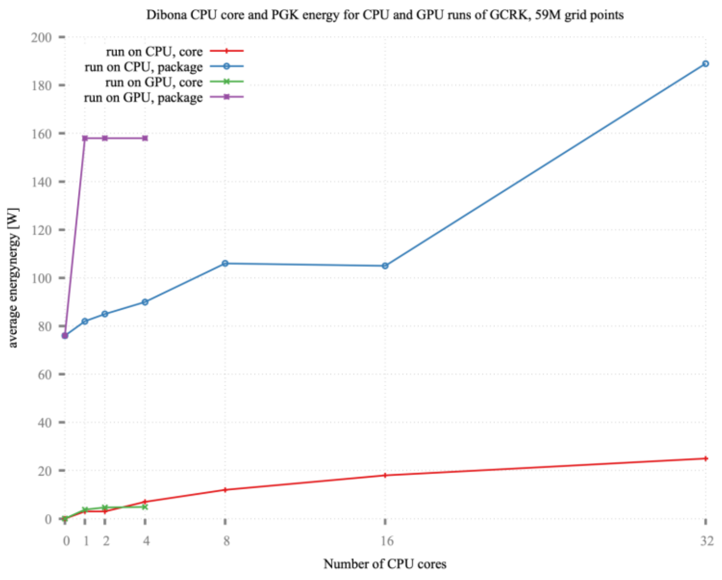


Figure 67 GCRK CPU power consumption when running on GPUs

Because the power consumption characteristic is the same for every GPU involved, we may expect the energy consumption to be constant for the fixed problem size (strong scaling) and increasing for the same size problem per GPU (weak scaling). In Figure 68, we correlate execution time to energy consumption for a strong and weak scalability. The top graph presents results for a fixed problem size (59M of grid points) and 500 iterations, and the bottom one for a constant problem size for each GPU accelerator (15M of grid points, 100 iterations). In both graphs, the blue line corresponds to exection time, which is plotted against left y-axis. The red line indicates energy consumption (in Joules), which is plotted against right y-axis. While the weak scaling allows to solve larger problems within more or less the same amount of time (problem size is fixed per accelerator), the energy consumption naturally increases by the number of accelerators being used. For strong scaling the situation is quite the opposite – increasing number of accelerators used results in shorter execution time and in lower energy consumption. We can achieve energy scalability and solve larger problems with almost constant energy consumption.

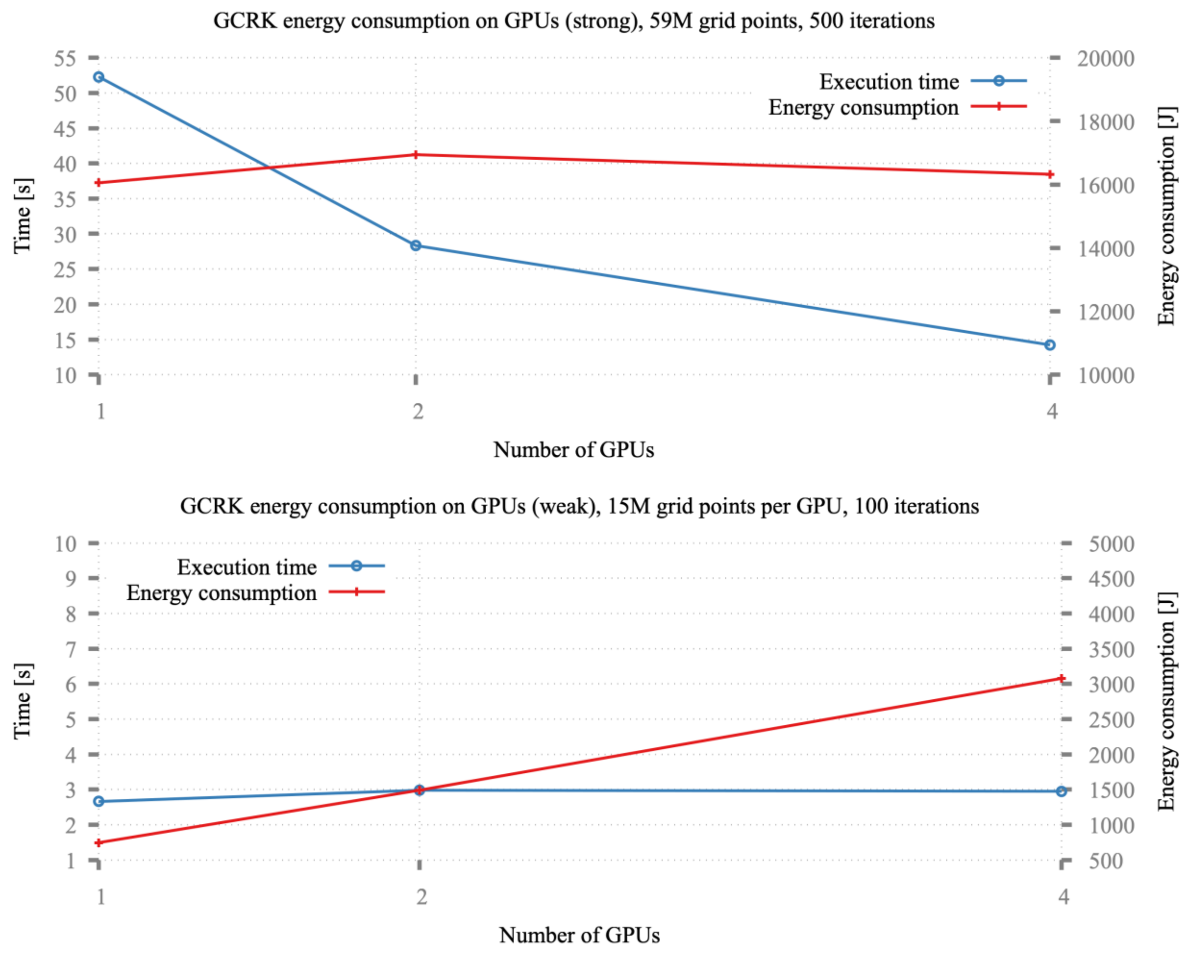


Figure 68 GCRK energy consumption on GPUs, strong (top) vs. weak (bottom) scaling

3.10.4 Measuring KPIs

In D6.1 we proposed KPIs to be measured for the UrbanAir kernels, which are reminded in **Errore. L'origine riferimento non è stata trovata.**.

KPI for computational efficiency	KPI for energy
- execution time - (strong and weak) speedup - number of iterations/time per iteration	- iterations/Watt

Table 22 UrbanAir KPIs

Let’s discuss how execution on CPUs compares to those on GPUs. For this test, we ran GCRK 100 iterations on a 59M grid points domain. The CPU code was run on multinode Altair environment, while Dibona was used for measurements on GPUS, as only these cards make it possible to run on such large domain. Figure 69 presents the results in terms of number of iterations per second. While increasing the number of nodes makes more iterations to be computed per second, the GPUs easily outperforms CPU version of the code. 64 CPU nodes are required to beat a single GPU, while beating 4 may be challenging. And even if it is doable, the energy consumption for such CPU run will be much higher than on GPUs.

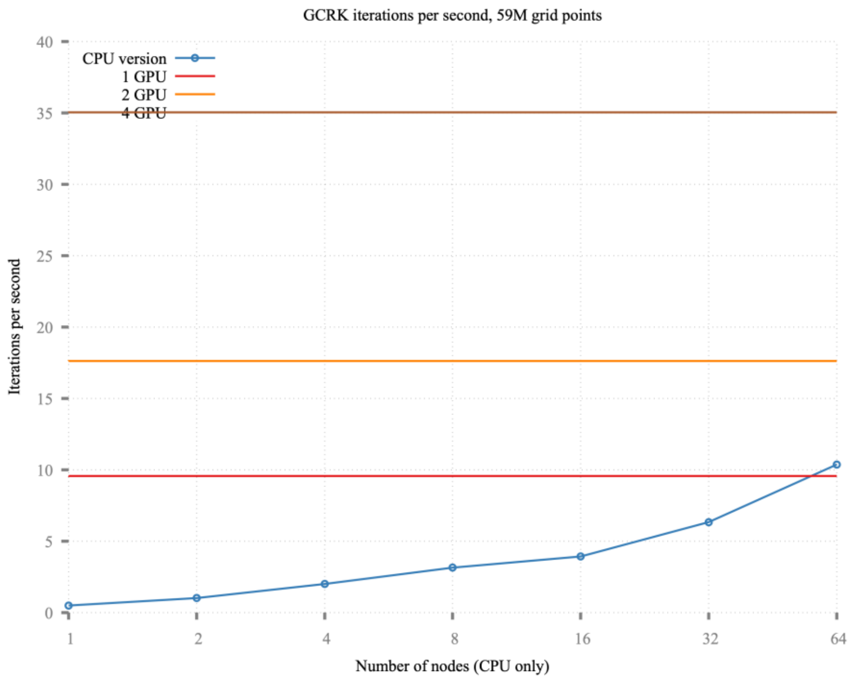


Figure 69 GCRK strong scalability iterations per second

Figure 70 presents the results for iterations per second for different hardware architectures, where the same amount of work is computed by each CPU or accelerator. As expected, the GPUs perform better than CPU, but also the newer GPU cards (Dibona) demonstrate its advantage over the older ones (Altair), even in the case GPU is assigned with rather small problem size.

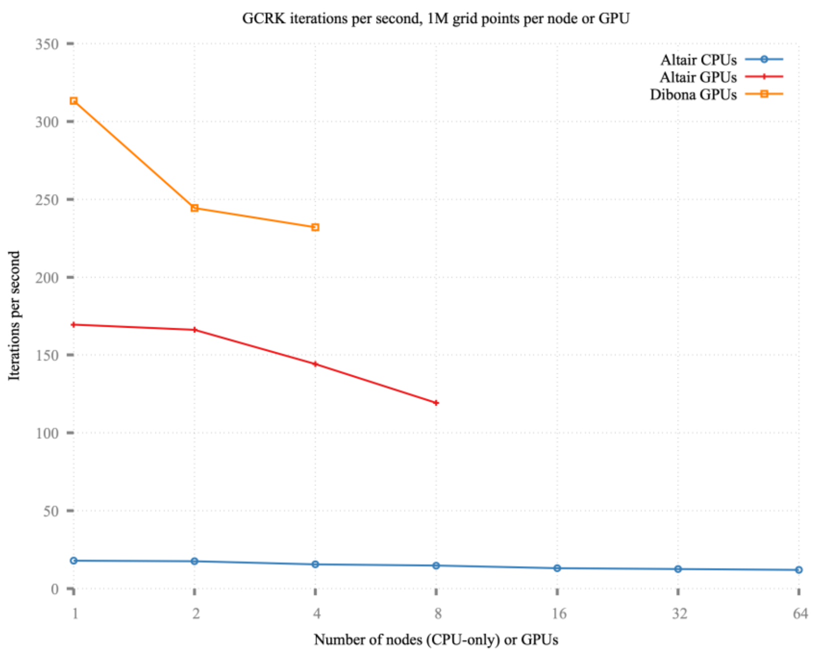


Figure 70 GCRK weak scalability iterations per second

Figure 71 presents iterations per Watt for strong (top chart) and weak (bottom chart) size problem. The energy measurements were done only on Dibona, thus we compare OpenMP parallelization to GPU accelerators. Similarly to the performance measurements, the iterations per Watt KPI is better for GPUs. However, when more GPUs are used to solve the problem, this KPI slightly increases for strong-scalability problem, while decreases for a weak-scalability.

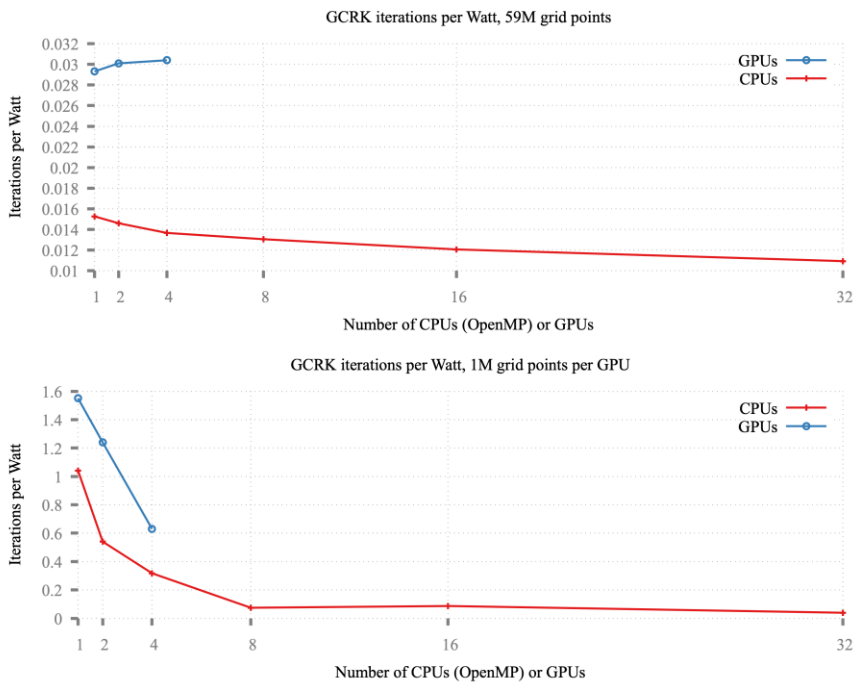
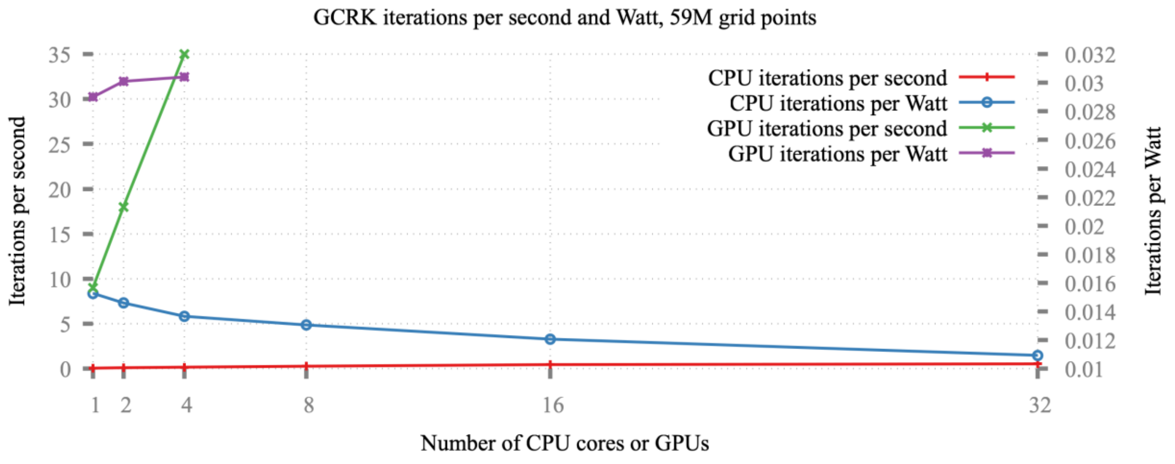


Figure 71 GCRK iterations/Watt, strong (top) and weak (bottom) scalability

Figure 72 presents cumulative KPIs measurements for different hardware architectures: CPUs and GPUs on a single node. Iterations per second are plotted against left y-axis, while iterations per Watt are depicted against right y-axis. For strong scalability, adding more CPU cores slightly increases iterations/s but significantly decreases iterations per Watt. Increasing number of GPUs significantly increases iterations/s keeping iterations per Watt at more or less constant level. It means that for GPUs, within the same amount of energy consumed we are able to perform work quicker.
For weak scalability, adding more CPUs results in both KPIs decrease. When more GPUs are used, iterations per second remains more or less constant, but iterations per Watt drops significantly.



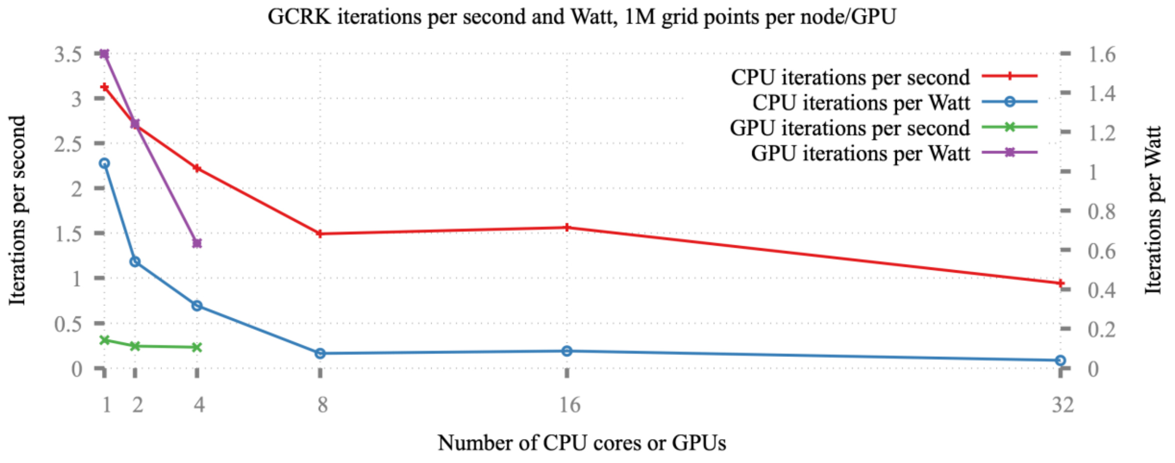


Figure 72 GCRK KPIs on a single node for CPUs and GPUs for strong (top) and weak (bottom) scalability

4 Summary and future work

In this deliverable we presented initial application benchmarks and results with respect to the KPIs defined in previous D6.1 deliverable. Each of the use case is progressing well, contributing to the overall project objectives, and benefitting from either adaption to heterogeneous resources, usage of mixed-precision or application of dynamic runtime systems. The first results are promising and demonstrate we are heading in a good direction. Smart cities were tested on several HPC platforms using ARM and Intel GPPs, with and without accelerators, where some of them supporting mixed-precision. CNR proposed some new algorithms and achieve already high performance and energy efficiency at the node level by exploitation of hybrid programming models for heterogeneous architectures and large scalability. RTM is extended with different formats of reduced precisions, providing stability tests and tests on image quality. RAIDER achieved O(10) improvements in energy efficiency and 2-3x in throughput, demonstrating using of APEIRON framework and intra/inter-FPGA communication IP. NEST-GPU reported increase in energy efficiency and reduced runtimes. HEP reported increase in energy efficiency, performance and throughput. TNM demonstrated increase in energy efficiency. INRIA created a new scheduler within StarPU and tested its robustiness and flexilbity usings ScalFMM and Chameleon applications. UrbanAir kernels achieved 3.5-9x speedup (multinode environment) and 2x increase in energy efficiency (single node).

We continue to work on the applications for further exploration how to increase energy efficiency and sustained application performance in particular. We do look forward to the availability of the ultimate IDV-A and application of other hardware solutions to test if and how this can lead to increase in our KPIs values. With the comprehensive set of use cases, applicable in different scientific fields, we will be able to draw conclusions on best approaches in increasing energy efficiency and sustained application performance and share with the community at the end of the project.

Next subsections detail future work for each application.

4.1 Smart Cities - CINI-UNIPI

The application and benchmarking activity of CINI-UNIPI will be extended to consider also RISC-V architectures and compare them to ARM-based (Neoverse N1, A64FX) and Intel-based GPP architectures. To this aim, COTS RISC-V solutions will be considered, at least scalar ones.

4.2 MathLib - CNR

Activities for the last year of the project include the implementation and testing of a new version of the BCMG linear solver based on a Communication-Avoiding Conjugate Gradient (CA-CG) method, based on grouping s-step of the standard CG method, so that latency and global synchronization are reduced. The final results will be hopefully obtained on the IDVA Textarossa platform. Therefore, the availability of the IDVA platform by the end of the 30th month of the project is a critical issue. It is worth to note that CNR group received a grant for early access to the Italian Leonardo supercomputer.

Therefore, as soon as it is available, some tests with our library will be run on the Leonardo hybrid supercomputer.

4.3 RTM – FRAUNHOFER

The next step will be the calculation of several shots to create an image more closely to typical RTM migrations. The image quality will be compared between float 32 bit and other compression formats. To make this feasible we will reduce the model to a 2D case which costs less compute power per shot.

4.4 HEP - INFN

The two identified benchmarks (CLUE and Pixeltrack) have been fully developed using the Alpaka library and will be portable without, or with minor, effort on the final IDV-A. In the next few months the development of this set of applications related to HEP can be subject to delays because there is a chance that the person working on this activity could be diverted to other projects at the CNAF INFN unit.

4.5 NEST-GPU - INFN

The preliminary comparison made between the multiprocess CPU version of the neural network simulation engine NEST (scaled up to what was available to run on either for ARM and x86_64 platforms with the Ampere Altra and the EPYC Zen2) and its GPU-supporting prototype NEST-GPU already reveals a staggering advantage of this latter – both in power consumption and runtime – even when run on just a single, recent but not latest generation device. Work is underway on the NEST-GPU application to move also the setup phase (which at the moment is still done on the CPU and represents a very unwieldy time at the start of every simulation) onto the GPU as well while the bulk of future work will focus on defining a scalable version of the testing protocol (likely to be based on the simulation of a-multi-area model of macaque vision-related cortex) and implementing it on an enhanced version of the NEST-GPU application in order to perform a sensible multiGPU performance measurement.

4.6 RAIDER - INFN

Preliminary results we reported in section 3.6 show that the FPGA-based implementation is the most convenient by a large amount when considering the energy efficiency aspect.
Our objective within the timeframe of the TEXTAROSSA project is, besides the mere technical development needed to finalize the processing pipeline by including the network interface with the RICH detector (data producer) and the trigger processor (results consumer), to scale-up performance of the RAIDER application to reach or surpass the challenging experimental requirement of a processing throughput higher than 10 MHz.
We will research along two parallel direction to reach this goal: i) exploit the intra/inter-FPGA scalability feature offered by the APEIRON framework, as shown up to a limited extent in section 3.6, and ii) continue improving the neural network model and the corresponding representation for input data used in the inference processing pipeline.

4.7 TNM - INFN

The TNM application plans to scale the simulation beyond the single-core baseline that we presented in section 3.7, both scaling on the number of CPU cores (using OpenMP) and accelerating parts of the code on NVIDIA GPUs. We also plan to add more sub-application as far as the HPC infrastructure allows, where we have other tools solving the Schrödinger equation or solving machine learning tasks with tensor networks. A scaling up to MPI is unlikely due to a serial step of matrix decompositions, which is necessary to obtain convergence.

4.8 MathLib - INRIA

Our next steps will be to develop FPGA kernels for ScalFMM and Chameleon, as well as to improve our scheduler to reduce energy consumption of the executions. For this latter point, we plan to replace the heuristics used to compute task priorities with those focused on minimizing energy usage, rather than just optimizing for makespan.

4.9 UrbanAir - PSNC

The next steps is to provide power consumption measurements on a multiple node to do a fair comparison between CPUs and GPUs and judge the increase in energy efficiency. We also plan to study if and how the mixed precision can increase the computational and energy efficiency.

5 References

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

[2] Bernaschi M, Celestini A, D'Ambra P, Vella F. Multi-GPU aggregation-based AMG preconditioner for iterative linear solvers, 2023. Available at arxiv.org/abs/2303.02352

[3] Nagasaka Y., Nukada A., Matsuoka S. High-Performance and memory-saving sparse general matrix-matrix multiplication for Nvidia Pascal GPU, IEEE 46th International Conference on Parallel Processing 2017, Bristol (UK), 101–110.

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

[5] 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.

[6] https://github.com/cms-patatrack/pixeltrack-standalone

[7] https://github.com/cms-patatrack/heterogeneous-clue

[8] https://www.khronos.org/sycl/

[9] https://indico.cern.ch/event/1106990/contributions/5096932/

[10] https://alpaka.readthedocs.io/en/latest/

[11] https://doi.org/10.3389/fninf.2022.883333

[12] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras

[13] Claudionor Coelho. 2019. QKeras. https://github.com/google/qkeras

[14] Duarte J, Han S, Harris P, Jindariani S, Kreinar E, Kreis B, Ngadiuba J, Pierini M, Rivera R, Tran N and Wu Z 2018 Journal of Instrumentation 13 P07027–P07027

[15] Vivado Design Suite User Guide High-Level Synthesis UG902 (v2020.1), Xilinx, San Jose, CA, 2021

[16] Topcuoglu, Haluk; Hariri, Salim; Wu, M. (2002). "Performance-effective and low-complexity task scheduling for heterogeneous computing". IEEE Transactions on Parallel and Distributed Systems. 13 (3): 260–274.

[17] Emmanuel Agullo, Berenger Bramas, Olivier Coulaud, Eric Darve, Matthias Messner, Toru Takahashi , Task-based Fmm for heterogeneous Architectures, Concurrency and Computation: Practice and Experience, Volume 28, Issue 9, 25 June 2016, Pages 2608-2629

[18] Clément Flint, Ludovic Pailla, Bérenger Bramas, Automated prioritizing heuristics for parallel task graph scheduling in heterogeneous computing, in PeerJ CS, 2021

[19] M. Ciżnicki, M. Kulczewski, P. Kopta, K. Kurowski, Methods to Load Balance a GCR Pressure Solver Using a Stencil Framework on Multi- and Many-Core Architectures, Scientific Programming, 2015