



D5.3 “Final report on development and optimization of use-cases and integration”

Version 1.1

Document Information

Contract Number	780681
Project Website	https://legato-project.eu/
Contractual Deadline	30. November 2020
Dissemination Level	Public
Nature	Report
Author	Micha vor dem Berge (CHR)
Contributors	Nils Kucza (UBI), Erik Funke (CHR), Dirk Michels (CHR), Felix Wiesenmüller (CHR), Raúl de la Cruz (BSC), Xavier Martorell (BSC), Sigrun May (HZI), Amani Al-Mekhlafi (HZI), Frank Klawonn (HZI), Hans Salomonsson (MIS), Daniel Ödman (MIS), Tobias Becker (MAX)
Reviewers	Amani Al-Mekhlafi (HZI), Leonardo Bautista Gomez (BSC), Jens Hagemeyer (UBI)

The LEGaTO project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Grant Agreement No 780681

Changelog

Version	Date	Description of Change
vo.1	29.08.2020	Initial ToC, based on D5.2
vo.2	27.10.2020	First input for chapters 2, 3 and 5
vo.3	05.11.2020	First input for chapters 4, 6, 7 and 8 merged
vo.4	06.11.2020	Merged version with updates in chapters 3 and 4
vo.5	09.11.2020	Updates of chapters 5, 6, 7
vo.6	13.11.2020	Internal review ready
vo.7	25.11.2020	Updates of all chapters, working on the internal review comments
vo.8	27.11.2020	Pre-final version, merging all updates from partners
v1.0	30.11.2020	Final version
v1.1	05.02.2021	Fix of broken link, add table of metrics to conclusion, harmonise metrics in use cases, add components overview to smart city, add metrics to executive summary, add Chapter 4.6

Index

1	Executive Summary	5
2	Introduction	7
3	Smart Home.....	9
3.1	Hardware evaluations and performance analysis with YOLO and Darknet	10
3.2	Smart Mirror Prototype Hardware Setups	11
3.3	Smart Mirror Software Optimization Progress	12
3.4	Baseline Benchmark and Performance	14
3.5	Development of additional Features	15
4	Smart City	21
4.1	Metrics & Optimization Goals.....	23
4.2	Baseline Benchmark.....	23
4.3	Development Status & Optimization Path	25
4.4	Final numerical results.....	27
4.5	A low energy-consumption Alya-oriented cluster	29
4.6	Increase robustness for Alya at large scale	30
5	Infection Research.....	31
5.1	Description of the accelerated biomarker discovery workflow	31
5.2	Metrics & Optimization Goals.....	35
5.3	Benchmarks	35
6	Machine Learning.....	40
6.1	Metrics & Optimization Goals.....	40
6.2	Optimisation	40
6.3	Integration	41
6.4	Experiments Setup.....	42
6.5	Results	43
7	Secure IoT Gateway	47
7.1	Components.....	48
7.2	Integration in the Smart Home use-case	55
7.3	Benchmarks	57
8	Overall project integration	62
8.1	OmpSs and XiTAO targeting SMP tasking.....	62
8.2	OmpSs with support for CUDA and OpenCL kernels	64
8.3	OmpSs with support for Xilinx FPGAs and DFIant kernels	66

8.4	OmpSs with support for Maxeler DFEs	68
8.5	OmpSs with support for Secure SGX tasks	70
8.6	Lint tool for OmpSs.....	72
8.7	IDE plugin for OmpSs	72
8.8	RECS Master and Slurm	73
9	Conclusion	75
10	References	77

1 Executive Summary

The main focus of Workpackage 5 is the development and optimisation of different real use cases with the help of the LEGaTO workflow. The final development and optimisation status is reported within this deliverable. There are five different use cases

1. Smart Home
2. Smart City
3. Infection Research
4. Machine Learning
5. Secure IoT Gateway

All of them except the latter have been optimised using one of the toolflows that the LEGaTO project provides. OmpSs was extended to make use of additional compilers and runtimes as described in chapter 8, so the use cases didn't have to implement all of them in a different way.

The Smart Home use case concentrates on a Smart Mirror demonstrator of which three versions have been developed within the project. A first demonstrator was built, after that, a second enhanced demonstrator was built with many manual improvements that have been carried out, also better standard hardware was used. Finally, the third demonstrator was built, based on the developed LEGaTO edge server, increasing the energy efficiency (measured in FPS/Watt) by 12x.

The Smart City use case simulates the air quality in urban areas. Existing code has been ported from Fortran to C, now running on x86 as well as ARM64 architectures. An implementation on an FPGA has been developed, showing an energy efficiency increase by a factor of 9, measured in GFLOPS/Watt. A further implementation on an Nvidia Xavier microserver was developed, increasing the energy efficiency by 95x (GFLOPS/Watt) at a cost of an 11x slowdown (wall time in seconds), compared to a BSC Marenostrum4 node. A full-size simulation run takes a total amount of ~1.3 kWh of energy, but as it runs multiple times a day with new input data, there is a lot of optimisation potential.

The Infection Research use case uses statistical methods to research on the effectiveness of drugs, vaccination strategies and harmfulness of pathogens. The major problem of this use case are the extremely long runtimes. To reduce this, four major compute kernels were identified and optimised, resulting in a wall time (in seconds) speedup of 10x, 40x, 544x and about 2503.6x for these kernels. Also, the total energy consumption in Joule for each compute kernel was measured and compared, resulting in a maximum energy efficiency increase of 7708.8x. A full run with a real dataset using the initial version of the biomarker candidate subset selection would take more than 1.5 years and consume around 187 kWh, so there is a huge potential in both, speed and energy efficiency improvements.

The Machine Learning use case on the one side develops neural networks with a focus on automotive usage, but also optimises existing neural networks with a new kind of deep learning optimisation tool called EmbeDL. With the help of EmbeDL, well known neural networks could be optimised to reach an average speed up of 4.3x (FPS) and energy efficiency increase by 6.3x (measured in inferences per Joule).

The Secure IoT Gateway makes it easy to secure many types of network connections to and from IoT devices. It bases on well-established technologies like OpenVPN, OPNsense and OpenWrt and adds an easy to use configuration Web GUI and rollout mechanisms on top of it. As analysis of the used components did not reveal any optimisation possibilities for computation speed or energy efficiency, this use case was used to enhance the security for the Smart Home use case.

Almost all targeted improvements in the six targeted disciplines energy efficiency, MTBF, code base security, designer productivity, TCO/costs and latency were reached, please see Table 17 in the Conclusion section.

2 Introduction

This document presents the final state of the implementation, optimisation and measurements of improvements of the LEGaTO use cases, representing the work carried out in Work Package 5 as shown in Figure 1.

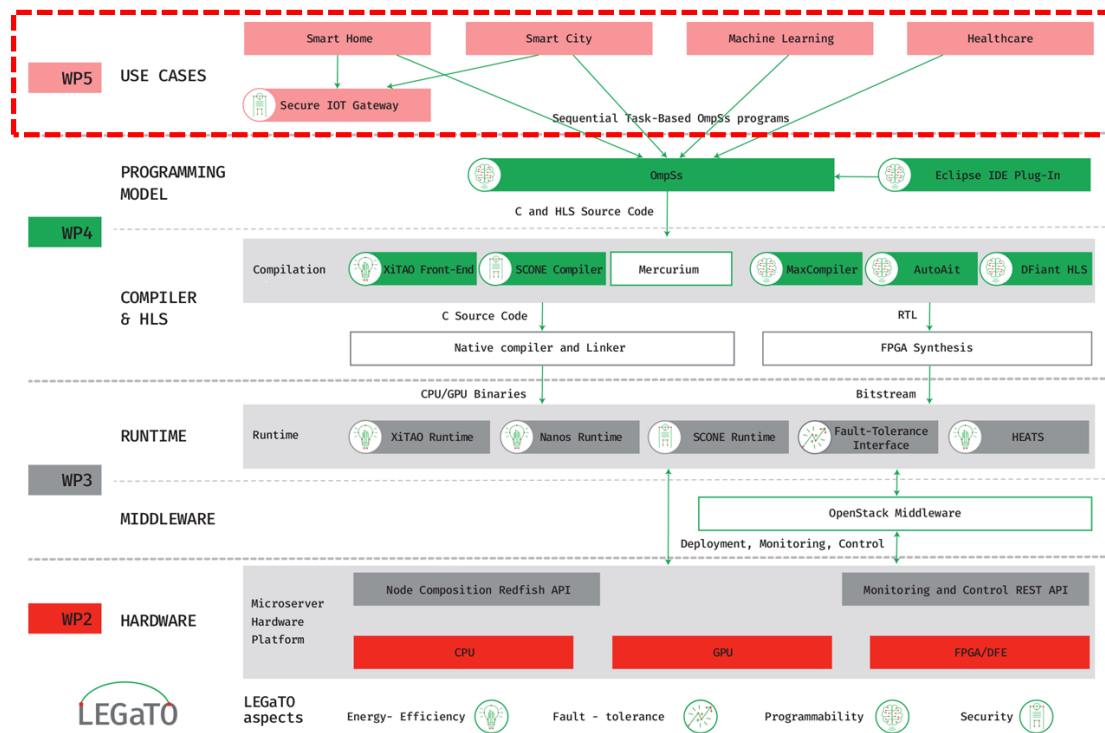


Figure 1: WP5 and the LEGaTO stack

In general, the four use cases of Smart Home, Smart City, Infection Research and Machine Learning have the goal to improve in terms of diverse metrics, such as energy efficiency, as mentioned below – while the Secure IoT Gateway focusses on adding extra security to other use cases. To improve on the diverse metrics, the use cases have been deeply analysed for their requirements and computational characteristics as presented in the first deliverable D2.1 [1]. Based on that, in D5.2 [2] baseline measurements and first optimisations of the use cases were presented, most of them could be massively improved and extended in this document. This deliverable is accompanied by D5.4 [3], the “Report on evaluation of efficiency and TCO improvements of use-cases” in which a detailed analysis of the efficiency and TCO improvements of all use cases is given.

Some use cases have been firstly optimised by hand, e.g. by porting it to a different programming language or by improving the algorithms itself. These optimisations showed already good results, e.g. in the Smart Mirror demonstrator of the Smart Home use case. As a second step, the already hand optimised algorithms were then improved by using one of LEGaTO’s main programming models, compilers and runtimes: OmpSs, XiTAO, DFiant, MaxJ as well as SCONe.

The optimisation targets of LEGaTO that we are targeting for are the following, as already depicted in D2.1 and D5.2:

1. Improve Energy Efficiency
2. Increase MTBF (by 5x)
3. Increase code base security (by 10x)
4. Increase designer productivity (by 5x)

In addition, we also consider the following objectives. These are not explicitly stated as objective in the DoA but are also relevant:

5. Reduce TCO/costs
6. Reduce Latency

The use cases were optimised using these programming models, compilers, runtimes and optimisation objectives:

Programming Model	Smart Home	Smart City	Machine Learning	Infection Research
OmpSs, XiTao, DFiant	1,4,5,6	1,2,4,5	1,4,5,6	1,4,5
MaxJ				1,4,5
SCONE				3

Table 1: Implemented optimisations and used programming models for the use cases

The structure of this document is quite simple, as every use case has its own chapter. In each chapter, a short introduction of the specific use case is given, followed by description of the optimisation process and measurements to show the improvements.

In case of the Secure IoT Gateway, the four main components and their features are described, as well as some benchmarks under different conditions and loads. Furthermore, the integration into two different locations of the Smart Home use case is described.

An important part of the project was the overall integration, showing that the used and improved LEGaTO frameworks could be used in the use cases, and are making use of the existing and newly developed heterogeneous hardware. This overall integration work is described in chapter 8.

The document finalises with a conclusion.

3 Smart Home

Current smart living environments are based on the simple automation of subsystems consisting of sensors, information processing, and actuators. New approaches are mainly driven by large enterprises, pushing big-data approaches, collecting as much information about the user as possible to derive the current action, and anticipating future behaviour. The development of assisted living can be seen as a move from isolated applications realized as simple embedded systems, towards cyber-physical systems gathering and processing large amounts of data from a high number of distributed smart devices. Additionally, the smart home must process interaction of different users simultaneously; conflicting actions have to be recognized (e.g., one user opening a window and another one closing it again) and compromises can be suggested. Different interaction schemes can be combined adaptively, e.g., switching from touch to speech interaction while cooking or using text-based output while phoning. Providing this functionality is highly computationally intensive. Since the collected data contains personal and highly sensitive information, cloud-based processing is undesirable. To address these privacy issues, we target resource-efficient edge computing.

In smart home environments, the smart mirror is a more and more frequently used interface for interaction. It is based on a display with a semi-transparent foil applied on it. This device shows personalized information and enables controlling of other smart components and services, e.g., operating the automated wardrobe, turning on/off the lights or opening/closing the entrance door. For this use case a demonstrator based on the open source project MagicMirror² [4] is developed and extended with the most needed features of smart homes. These include face recognition, object recognition as well as voice and gesture control. All developed modules are published open-source on GitHub [5].

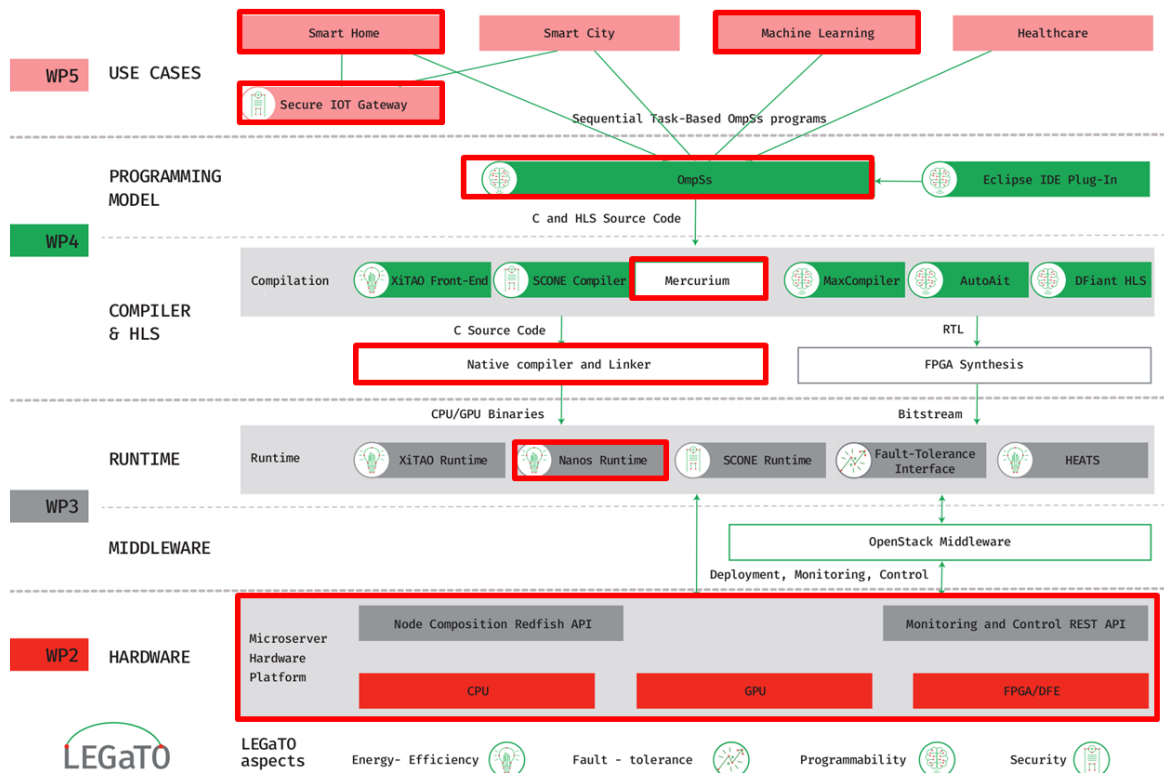


Figure 2: Used Tools in the smart home use case of the LEGaTO stack

Figure 2 shows the LEGaTO Tools used in this use-case. These are OmpSs with the Mercurium compiler and the Nanos environment and the hardware developed in this project. With this the energy efficiency is increased by factor 12 and compared with other implementation frameworks (MPI) the designer productivity was greatly increased.

3.1 Hardware evaluations and performance analysis with YOLO and Darknet

Multiple features of the smart mirror are based on the neural network called YOLO. It is a commonly used network for object detection. In the mirror application, two instances of this network recognize around 80 objects and 32 hand gestures. Therefore, it can be used as an indication for possible performance and hardware requirements. In this chapter, the performance on different hardware architectures is evaluated in regards to FPS and power consumption of running a single instant of YOLO for object detection. This is visualized in the following graph (see Figure 3) for NVidia TX2, Xavier embedded modules and GTX 1080Ti or RTX 2070 GPUs. In the prototype used Intel RealSense supports a maximum framerate of 30 FPS and is therefore marked by the red line. To measure the maximum performance on the GPUs, a video was used. The GTX 1080Ti runs one YOLO instance with 30 FPS and 160W power consumption with the RealSense being the bottleneck. With a video around 54 FPS are reached, but the power consumption is increased to 210W. This is way too much performance and power consumption for an embedded hardware solution. The introduction of tensor cores on the RTX 2070 shows a possible solution for this. While bound by the RealSense 30 FPS about 90 W are reached. The maximum performance of one YOLO instance is 64 FPS and a power consumption of 128 W. If the tensor cores are not utilized, the maximum performance is reduced by 10 FPS, and the power consumption is increased by around 30 W. This demonstrates the potential of specialized hardware accelerators for neural networks. For possible embedded hardware solutions, NVidia TX2 modules are evaluated. With 13 W and 4 FPS the performance is not sufficient for the smart mirror use case. The NVidia Xavier module on the other hand shows a decent performance of 24 FPS at a power consumption of around 40 W in the max performance mode. The integrated tensor cores of these modules leveraged a possible embedded hardware solution.

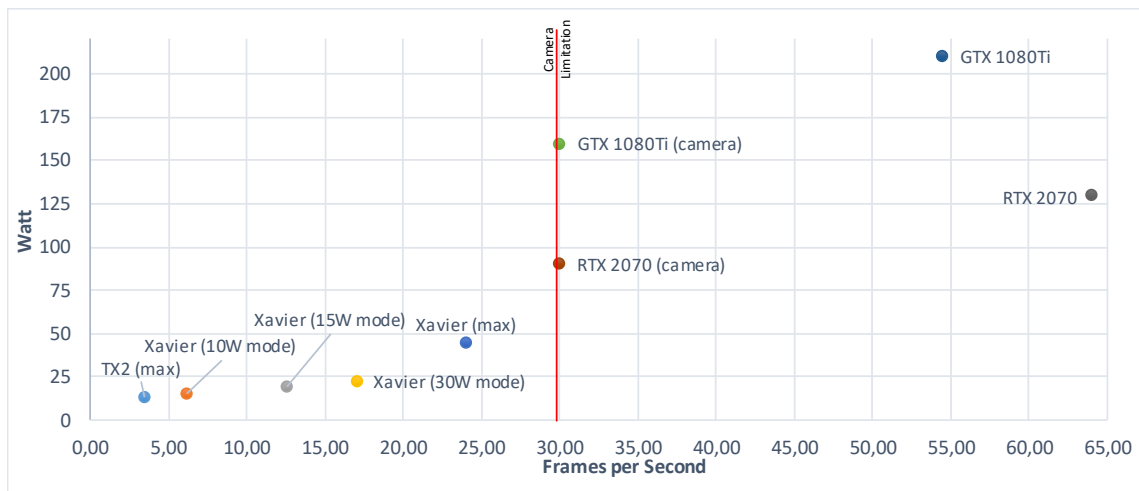


Figure 3: FPS and Watt of Yolo running on TX2, Xavier, GTX 1080Ti and RTX 2070. The application is limited by the maximum output framerate (30 fps) of the Intel RealSense d435i (red line).

3.2 Smart Mirror Prototype Hardware Setups

The smart mirror prototype is composed of a display with an applied semi-transparent mirror foil, a camera and a workstation. While first test was conducted with a small monitor and normal webcam, the later version of the demonstrator is utilizing an Intel RealSense and a big TV as a display. The RealSense was chosen for the featured depth images. These prototypes were also shown at several fairs to promote the LEGaTO project. At each workstation smart mirror prototypes, the overall application was distributed to two accessible GPUs, due to memory limitations on the GPU and to enhance the hardware utilization and user experience. On the embedded edge solution, complete processing parts are distributed over multiple microservers.

3.2.1 The First Prototype

The workstation for the first prototype was composed of two GeForce GTX 1080Ti GPUs and an Intel i7-7700K processor with 32 GB of RAM. On this setup the first version of the demonstration was running with around 12 FPS in all detections simultaneously (face, gestures and objects) and a power consumption of 650W. After porting the limiting python scripts to C or C++ with CUDA optimizations, especially for Intel RealSense camera handler, the performance was enhanced to 16 FPS for the three detections. This can be traced back to a high CPU computation, for example, due to image scaling in python. Tracking and other additional features were not yet implemented in this setup. After the evaluation of the RTX 2070 the first setup was discontinued for the last period.

3.2.2 The Second Prototype

The second prototype is composed of two GeForce RTX 2070 and an Intel i9-9900K with 32 GB of RAM. The introduction of Tensor Cores increased the performance to around 20 FPS for all three detections (face, gestures and object). At the same time, the power consumption was decreased to around 430W for the workstation. After additional optimization of the communication structure between the modules within the smart mirror application (number of image streams, partitioning) the performance was increased to 25 FPS for all detections. The latest features, like tracking, are added on this setup with minor performance reductions.

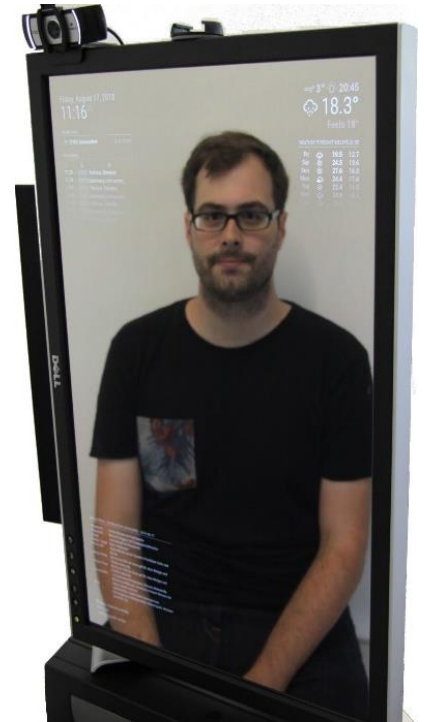


Figure 4: The first prototype setup



Figure 5: The second prototype setup

3.2.3 Embedded Hardware Prototype

As shown in the evaluation of the Darknet framework on different GPU hardware, several NVIDIA Xavier modules should be very well suited to run the mirror. The first embedded hardware setup consists of two AGX Xavier modules, which are interconnected via a PCI Express bridge. Therefore, one module is configured as an endpoint device and a virtual Ethernet connection is established. The bandwidth between those two modules is around 5 GBits/s with a roundtrip time of 1.5 ms. The Xavier modules are also one target architecture for the embedded edge server, but the performance of the virtual network within final edge server is expected to be much higher (around 40 GBits/s with a roundtrip time of < 1ms). Therefore, this setup can be seen as an intermediate step towards the final targeted hardware. The two modules have a combined maximum power consumption of 100W. The complete smart mirror application is ported to this setup, and the functionalities are partially optimized to utilise the second module. Hereby a performance of 16 FPS was achieved in the simultaneous execution of all detections at an energy consumption of 55W.



Figure 6: Dual Nvidia Xavier prototype setup

3.3 Smart Mirror Software Optimization Progress

To improve the user experience and increase performance/energy efficiency, the data flow between the modules was changed, modules were reconstructed, and Kalman filters were introduced. These changes have improved the performance and reduced the power consumptions of the smart mirror demonstrator. Starting from about 12 FPS in each detection at a power consumption of 650W, the second iteration reaches 25 FPS at a power consumption of 430W on workstation hardware while all detections run simultaneously. Additional features have been implemented that increase usability and developer friendliness, such as a decision-maker who calls applications and displays content based on all information. Further optimizations and porting for embedded hardware made the mirror demonstration running on a single Nvidia Xavier module. Due to the high number of neural networks and computations, this single device is capable of running everything

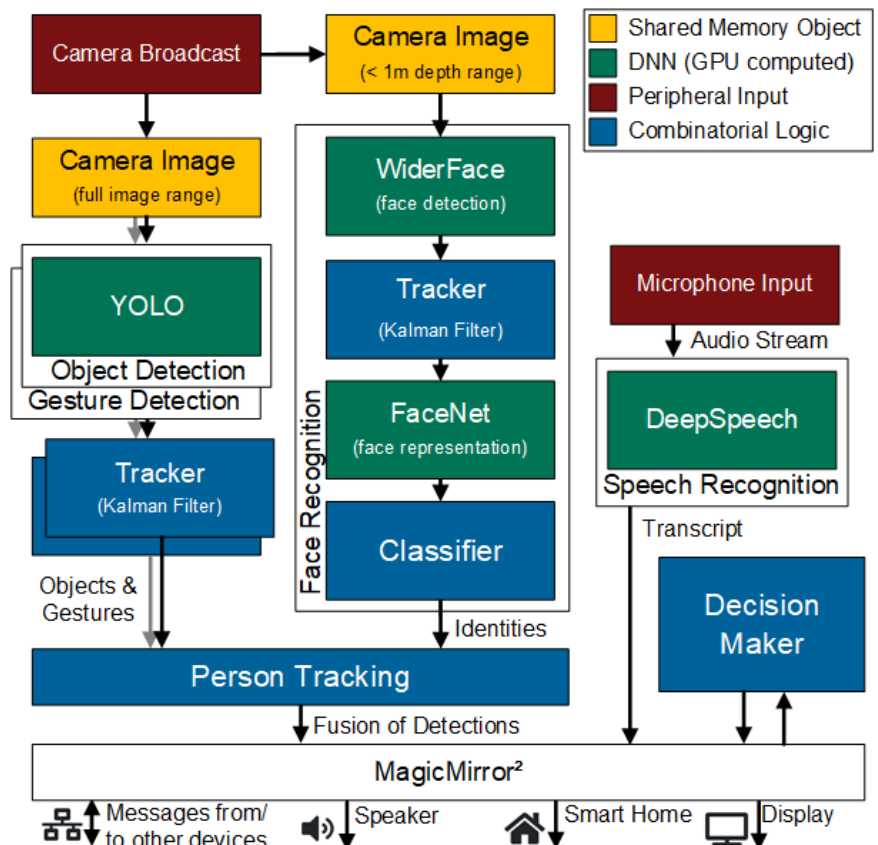


Figure 7: General structure of the smart mirror application

with 6 FPS by a power consumption of 42 W. The bottleneck in running on a single module is the GPU, which was fully utilized while core computation was free. With the help of OmpSs@Cluster, the computation of the object and gesture detection including the tracking is sourced out to the second Xavier. Thereby the GPU of the first module is greatly reduced by the cost of a slightly higher CPU utilization. This setup is running the full smart mirror with all detection at 16 FPS in average with a power consumption of 55W. With this setup the targeted performance in FPS per Watt is higher than the targeted result. The result of the final embedded edge server can be estimated to be similar, as the Xavier modules will also be used and consume the majority of the energy.

3.3.1 Module Reconstruction and Optimizations with C++ and Acceleration with CUDA

Several modules were first implemented in python because it is the common language for neural networks, and therefore a fast development could be done. An efficient image processing was neglected at first. After the initial evaluation, the following improvements were made.

- The main camera script, which prepares the RGB and the depth image, was ported to C++ and accelerations with CUDA were made. The CPU load has been reduced by 50% at the cost of a slightly increased graphics card load, which is designed for such image processing. This script provides the image to all other scripts via GStreamer appsink.
- To greatly reduce latency, the image showing all detections is created within the camera script and handed over to the Magic Mirror Framework. This workflow also has also removed the need to merge images of each module and removed the need to send back images from each module.
- Every neural network previously running on the smart mirror has needed a unique image resolution. In the first python version, this downscaling was also done by the CPU and before each inference and was thereby slow and inefficient. This was rewritten with GPU acceleration at a combined script. Some neural networks are also altered to use the same image resolution. Implementation via FPGA is desirable in the future.
- The darknet framework is also written in C and uses its own image format. The flow of scaling and converting is next to be combined and optimized.
- The first implementation of face recognition tried to identify each face in each image. This cost a lot of computation due to a large number of possible faces. By the introduction of the depth image this could already be reduced. By implementing tacking, faces do not have to be identified in every frame. In the current version, the identity is only checked once every second. This reduced the resource requirement enormously.

These changes together with the introduction of tensor cores have increased the performance from around 12 FPS at the first prototype to 25 FPS on the second prototype. They were also applied to the mirror running on a single Nvidia Xavier module but have shown only 6 FPS.

3.3.2 Offloading of detections with Darknet using OmpSs@Cluster

Using OmpSs@Cluster, the computation of the object and gesture recognition is shifted from the system running the smart mirror application to an additional module. These modules are coupled via a network infrastructure and are using MPI as a backbone. In our case two Nvidia Xavier modules are coupled over a virtual Ethernet via PCI Express (see section 3.2.3). The general structure is shown in Figure 8. If only the object detection is examined, just the image conversion from the image

source into the format of the darknet library and the publication of the results is remaining on the first module. With the help of OmpSs@Cluster, the computation of the DNN and the tracking algorithm is entirely moved to the second module. All steps in this flow are separated into tasks, so all blocks can be executed in parallel, and the communication time is hidden. If only the execution of a detection alone is observed, the usage of the GPU is completely outsourced to the second Xavier module and the utilization of the CPU is increased a little bit. The performance in FPS remains identical to running it on a single node and is bounded by the throughput of the camera (30 FPS). Since the complete mirror application requires too much computing power from one module, the performance could be increased from 6 FPS to 16 FPS in all detections simultaneously by shifting the object and gesture recognition to the second Xavier module. This has moved the bottleneck from the GPU of the first module to the CPU cores of the first module. Further outsourcing of CPU computations will increase the performance even more. A slightly higher power consumption due to the communication overhead, next to the increased power consumption due to having two Xavier modules, has to be tolerated. For a more detailed description, see Deliverable 3.4 section 4.2.3.3.

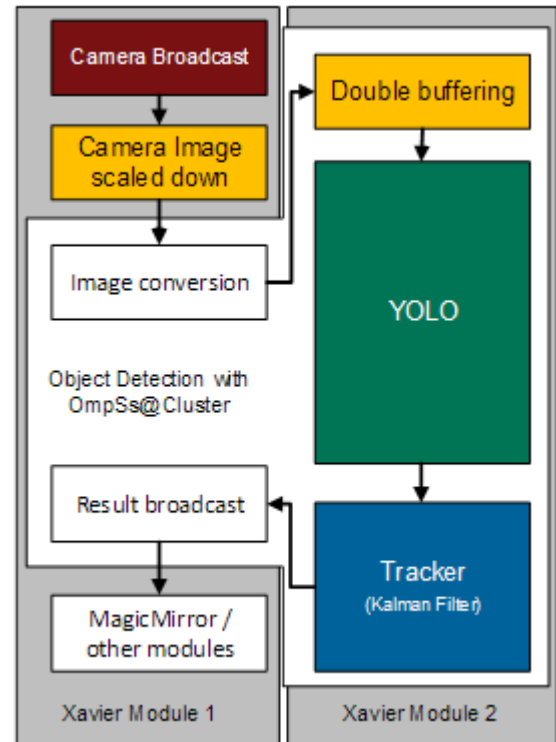


Figure 8: Structure of the offloaded task for object/gesture recognition using OmpSs@Cluster

3.4 Baseline Benchmark and Performance

As the baseline for the entire Smart Mirror, the total power consumption and the achieved frames per second in all detections simultaneously are considered. All results are shown in the following three graphs (see Figure 9). The colour of all three graphs represents the three different hardware setups and the goal. The first graph shows the achieved FPS, and the second graphs shows the average power consumption after each optimizations step. The third graph shows the combination of the previous graphs to express the performance. Here higher is better. The initial prototype achieved a performance of 12 frames with a power consumption of 650 W. This results in a value of around 0.018 FPS/W. After the first optimizations and translation from python to C++ with CUDA 16 FPS with a power consumption of 650W was achieved. This results in a value of around 0,024 FPS/W. With the second hardware prototype and the introduction of Tensor Cores, the performance was increased to 20 FPS with a reduced power consumption of 430 W, which corresponds to a value of around 0.046 FPS/W. After modification of the communication infrastructure and a finer granular subdivision, the performance was increased to 25 FPS with the same power consumption. Thereby around 0.058 FPS/W is achieved. The efficiency was thus increased by factor 3 due to the optimizations carried out so far. The optimized version was also runnable on a single Nvidia Xavier module. It achieved a performance of 6 FPS at a power consumption of 47 W. The Bottleneck, in this case, is the GPU, which is not capable of handling the high amount of DNNs. After offloading the computation of the object and gesture detection to the second Xavier module, the performance was increased to 16 FPS in all detection simultaneously with a power consumption of 55 W. This relates

in an efficiency of 0.29 FPS/W, which is higher than the goal of 0.2 FPS/W. If you compare the efficiency of the first prototype with the efficiency on the embedded prototype, the energy efficiency is increased by a factor of 12. This is possible by combining specialized hardware with the help of an adequate middleware.

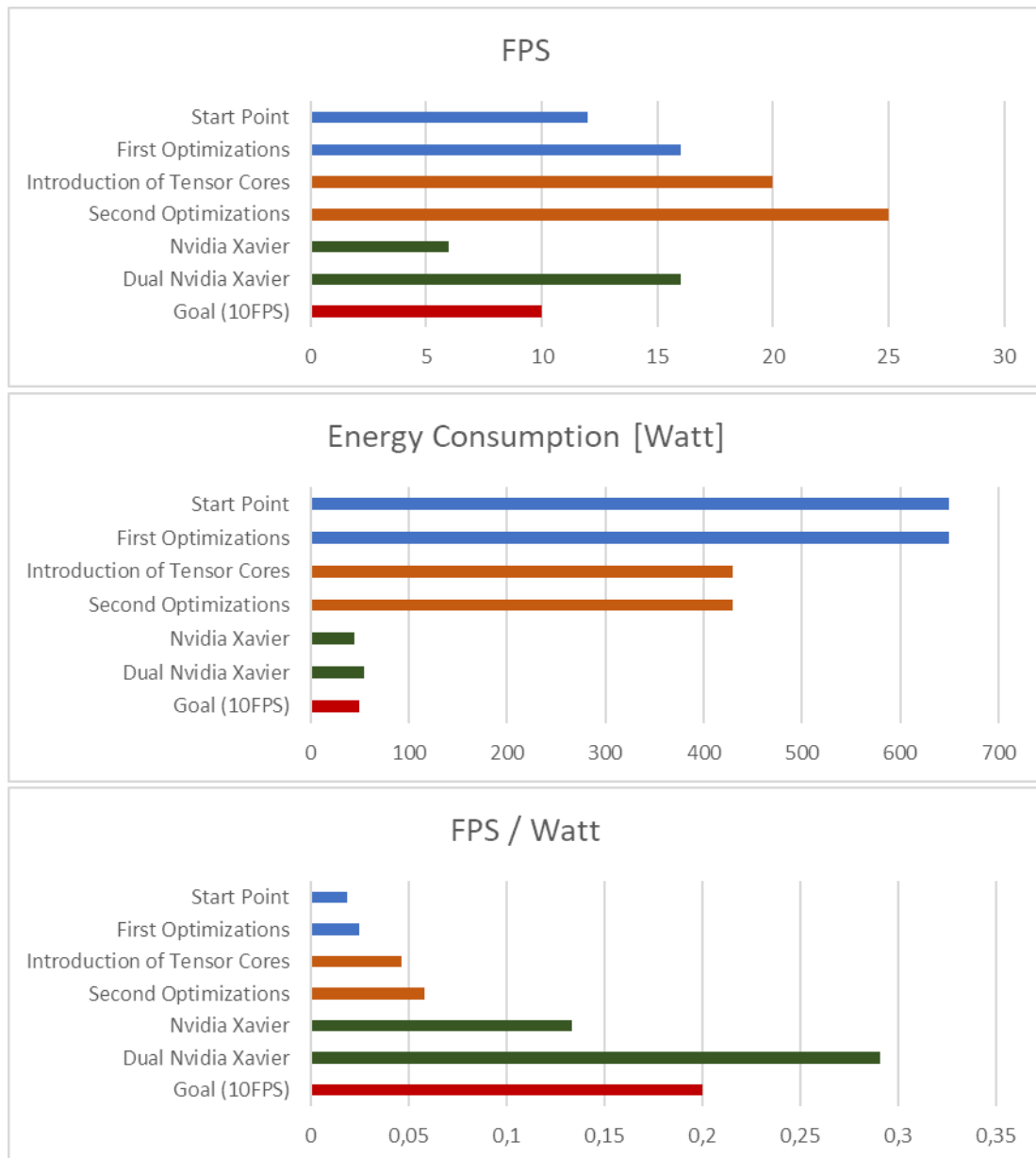


Figure 9: FPS, Watt and FPS per Watt comparison between all performed optimization steps. Marked in blue the first, in orange the second and in green the embedded prototype. Red is the goal of 10 FPS by 50 Watt.

3.5 Development of additional Features

In order to increase user-friendliness, additional functions are being developed. These include hand gesture recognition to control the mirror, Kalman filter and Hungarian algorithm for tracking detections, user behaviour prediction and recommendation. Therefore, a dataset of 32 hand gestures and a dataset of user interaction of the mirror is created.

3.5.1 Implementation of Tracking using Kalman Filters and Hungarian Algorithm

A very important feature is tracking the recognized faces, objects, and gestures. This allows many simplifications and increases usability. In the beginning, all decisions were based on findings only, but now the tracked detections are used to make decisions. Kalman Filters are used to predict the next value for each detection. In our case, the input of these filters is the bounding boxes of each detection in each frame. With the Hungarian algorithm, the current detection and the predictions are cost-efficiently associated. After a small number of frames, an ID can be assigned for each detection. Thereby the current detection is related to the previous detections. Predictions of the Kalman filter are also done and assumed to be valid even if no detection is visible for a short time. All different tracked detections are associated, and the users can be tracked, even if they turn their face away, for example. The detection of persons is the key point for the whole mirror. For this approach, a Kalman filter is needed for each detection and cannot be shared. Many calculations can be done in parallel and are done so with the help of `OmpSs parallel for pragma`.

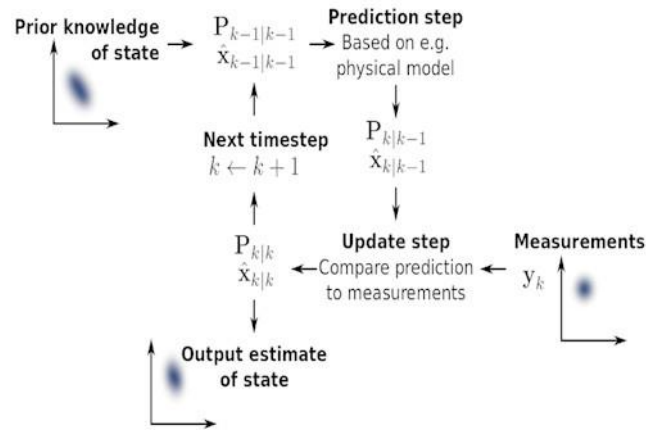


Figure 10: Workflow of Kalman filter and Hungarian algorithm for detection tracking

3.5.2 Dataset of Hand Gestures for easy control

For an easy to use control of the smart mirror, hand gestures are used. No freely available data record has existed for this scope, so a custom data set had to be created. In order to train a DNN to recognize the desired gestures, a huge dataset is needed. The result consists of 32 hand gestures of 13 persons (examples for gestures are shown in Figure 12). We distinguish between left and right hands and rotations are considered (e.g. thumbs up or down). This results in an amount of four hundred thousand images. Three persons are used for a test dataset, and nine people are in the training dataset. Further extensions are planned, and further special cases will be covered. Custom YOLO networks are trained with this dataset with the darknet framework. Some image augmentations are automatically done by this framework (e.g. colour shifts, scaling). The best results are achieved with yolov4-tiny with three YOLO layers. This runs on a NVidia Xavier with 30 FPS on the GPU by a utilization of 60%. The training reached a mean average precision of 92% by an intersection over the union threshold of 0.5. This is an

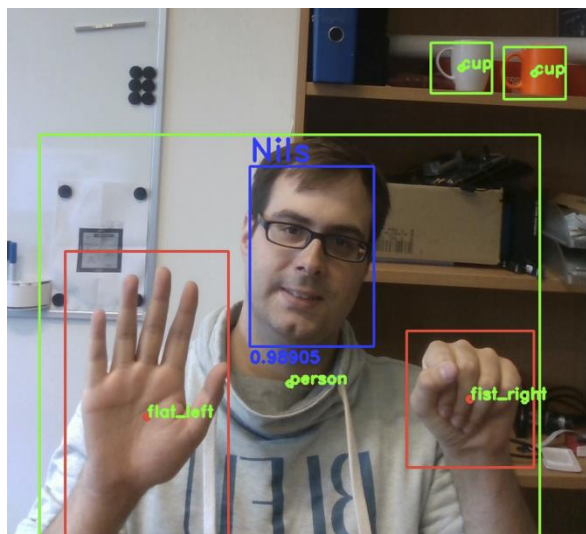


Figure 11: Gestures are used to control the main menu. This example shows all outputs of the DNNs. Gesture with the center of the gestures in red

extraordinary result considering the size and texture of hand gestures. Due to data privacy reasons, the dataset cannot be made publicly available. A publication of the trained network is planned after further revision. For this purpose, further neural networks will be trained and compared. Due to the current world situation and difficulties in data acquisition, this was not possible within the scope of LEGaTO.



Figure 12: Example for the hand gestures in the dataset

3.5.3 Behaviour Prediction of the User Interaction

For a really intelligent mirror, a form of behaviour prediction is mandatory. Therefore, a large amount of usage data is needed. In order to gather this data, all messages broadcasted by the modules within the mirror are parsed and the status of the mirror is stored within a database. This database is continuously enlarged while running the smart mirror. Figure 13 shows the pipeline of this. Due to the current situation only, a small dataset could be gathered so far and interaction with multiple persons is also hardly represented. The first prototype is permanently exhibited on the corridor of our work rooms and trained test persons stand in front of it at regular intervals. This

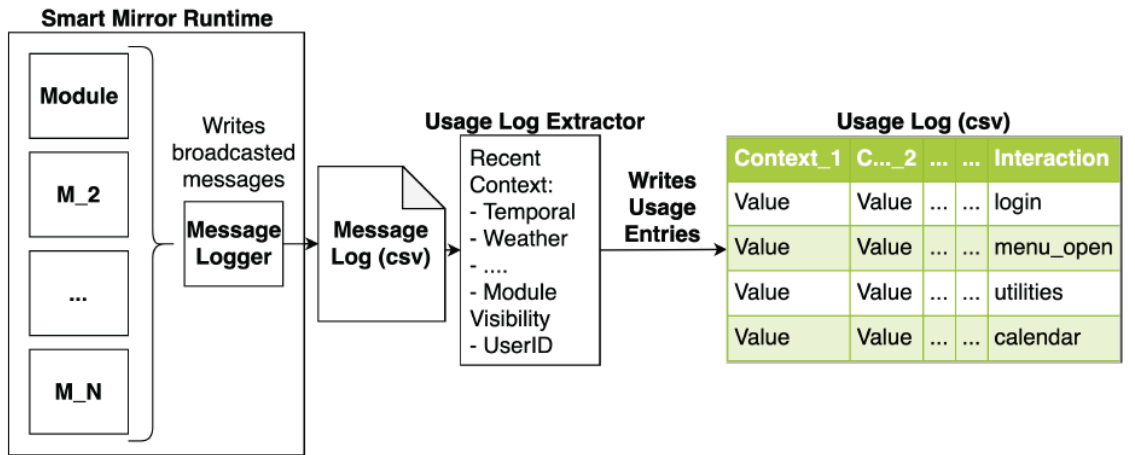


Figure 13: Pipeline for status recording. These data entries consist of the status of the whole mirror and all information of the installed modules (e.g. weather forecast, mensa offering, visible objects).

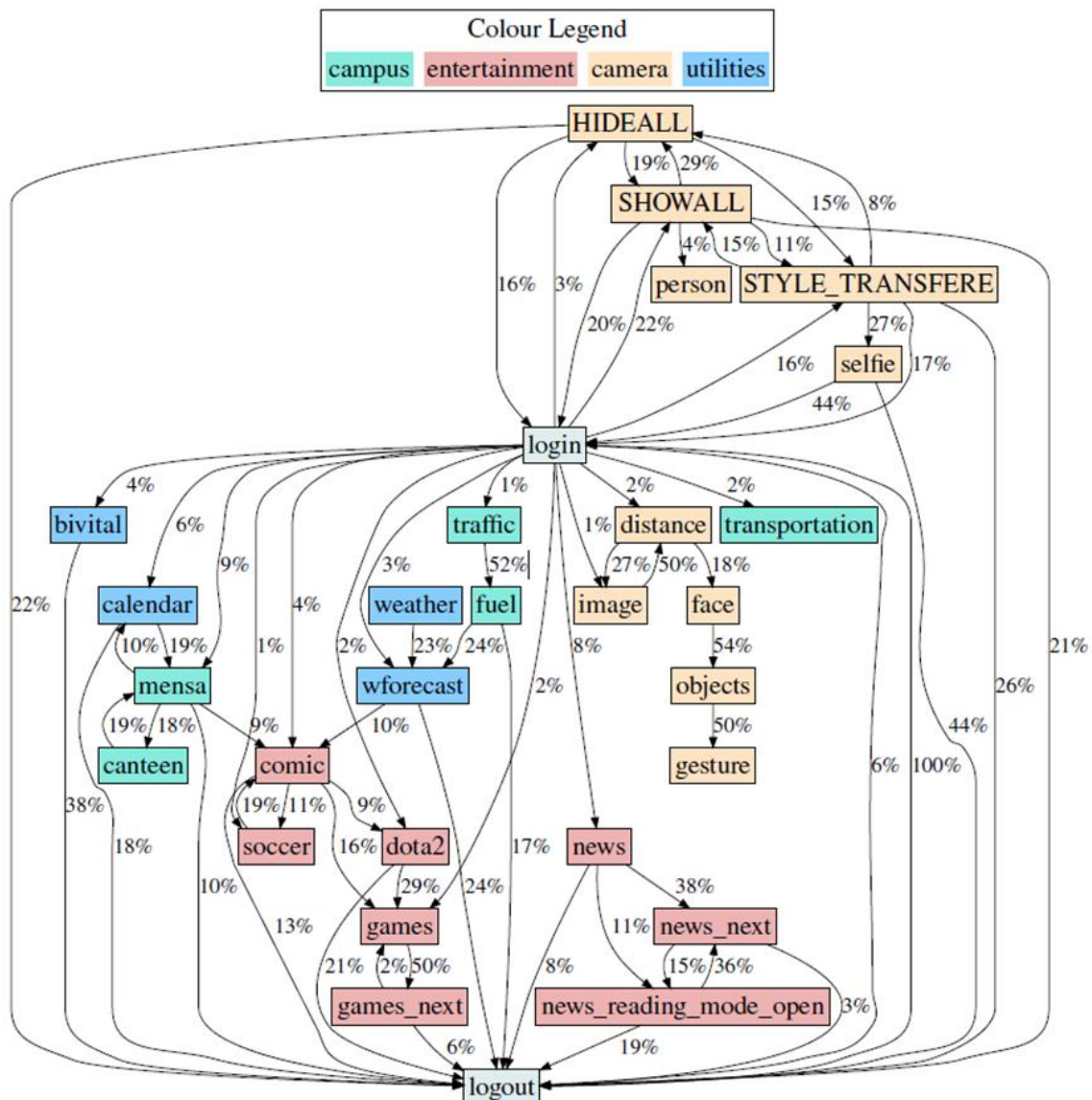


Figure 14: Visualisation of the gathered dataset. All points represent an action to open or modify the according modul. All actions can be grouped into 4 types. Campus, entertainment and utilities group the according modules and camera for the smart mirror

shows to be a good source of individual data sets. Sadly, public events, like fairs, were already virtual as we gathered data and due to distancing rules, data of presentations of the mirror are lacking. This must be considered in the following examinations and it explains why so far only the first examinations have taken place. The gathered dataset can be visualized with the graph in Figure 14. It shows the transition probabilities of all test users between the possible actions. The current status of the mirror offers a total of 33 different actions. The actions can be grouped into 4 different types. Campus, entertainment, and utilities are different categories of according applications (e.g. weather of utilities or cafeteria offering for campus). The category camera consists of all possible actions with regards to the camera of the mirror (e.g. show different detections or style transfer). The start point of all interaction with the mirror is the login state in the middle and all interactions stop with the logout of the user. Based on these data, different models were examined. The input for all subsequent models is the last 32 data entries of the database. Figure 15 shows the simple Markov chain representation of this graph. This is already a very good model to describe the problem of behaviour prediction for our case. Other investigated techniques of machine learning are: Adaptive Boosting classifier (ADB), Most Frequent Used (MFU), k-nearest neighbours (kNN),

smartphones). Unfortunately, these cannot be compared directly with the values achieved here because the initial situation and implementation are too different. For a first recommendation system, the previously mentioned networks already show a sufficiently high accuracy as long as it is not necessary to differentiate for all users. If a user stands in front of the mirror doing nothing for a long time and the prediction shows a certain accuracy, a proposal can be made. Here the behaviour of other users can also support. A further step can be to detect anomalies if the behaviour of one user is in total contrast to the user's normal behaviour.

4 Smart City

In many urban areas, air quality and associated impacts on public health are matters of growing concern. The emission and dispersion of critical pollutants (PM_{x_y} , NO_2 and ground-level O_3) correlate with cancer, asthma, cardiorespiratory problems, brain development in children and reduction of life expectancy in general [6]. As a consequence, air quality monitoring networks and modelling forecasting systems are critical to increase awareness and, ultimately, to assist decision-makers on the adoption of measures to protect public health.

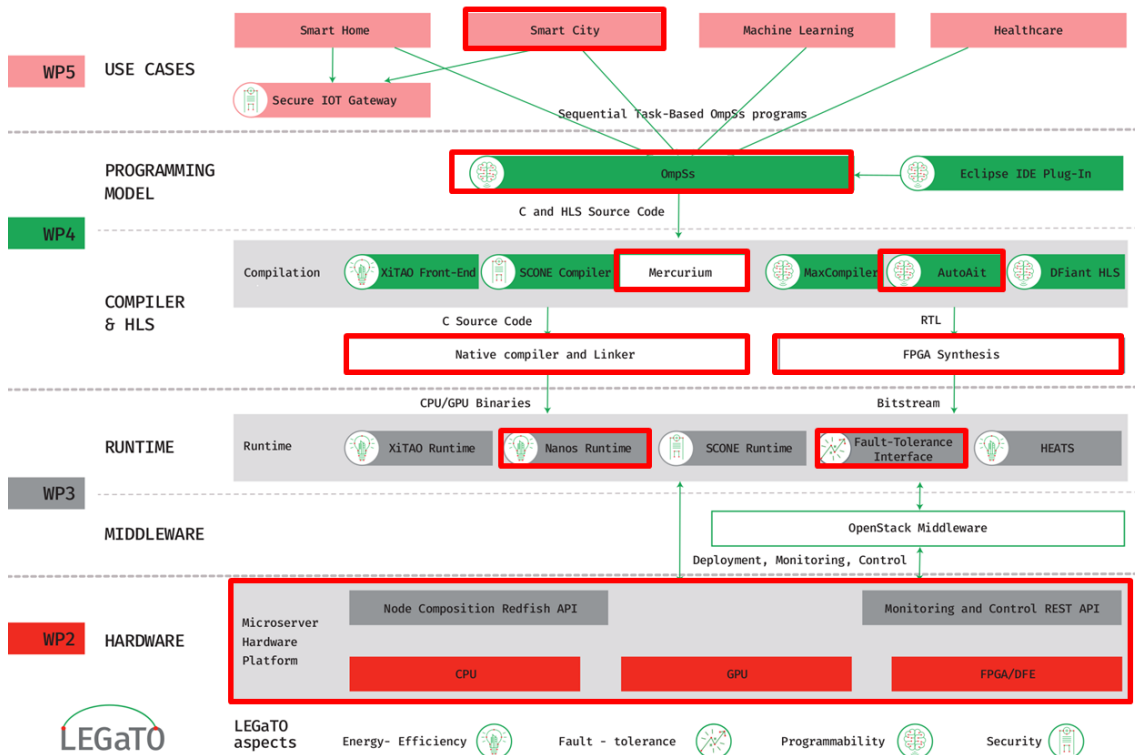


Figure 17: Used Tools in the smart city use case of the LEGaTO stack

The Smart City use case is composed of four different components that, individually, meet a specific project objective and, in combination, allows the development of an operational air quality modelling system at street-level resolution based on CFD. Figure 18 shows the key components of the air quality modelling system. Most of the individual components have already been developed within other research projects. In this operational workflow, different inputs are required by the CFD model to simulate urban-scale winds:

- **Meteo data:** CFD models require initial boundary conditions to confine the physical problem into a finite computational domain (e.g., a city mesh). These boundary conditions set the inputs of our CFD simulation, defining how the fluid, or wind in our case, enters (inlet) or leaves (outlet) the domain. In other words, boundary conditions connect the region of interest (our urban area) with its surroundings.
- **Sensor assimilation:** In order to model and forecast urban-scale pollutant dispersion, it is not only necessary to dispose of high-resolution near-surface wind fields, but also to characterize the sources of pollutants at street-level (mainly derived from vehicle combustion) through sensors or emission inventories

- **CFD-based wind and pollutant dispersal modelling:** Within the Smart City use case, the CFD-based simulator is the main core of the whole application. To this end, an urban-scale wind forecasting system was developed, based on coupling the meso-scale WRF model (a numerical weather prediction system) [7], with BSC's Alya modelling system [8], a CFD-based simulator. In order to simulate the air flow through the urban-scale morphologies (buildings), the CFD model solves the incompressible Navier-Stokes equations on a computational mesh that represents the city geometry. However, due to its physical complexity, the use of CFD techniques requires massive computing resources.
- **Data gathering and streaming:** After the simulation, data is collected directly from the CFD model and the environmental network. Pollutant data is post-processed and published using an Open Data format in a BSC repository. The post-processing may require large amount of memory resources to be interpolated and stored in databases depending upon the mesh size and refinement. Finally, the data provided might be used by visualization and analysis tools for their study.

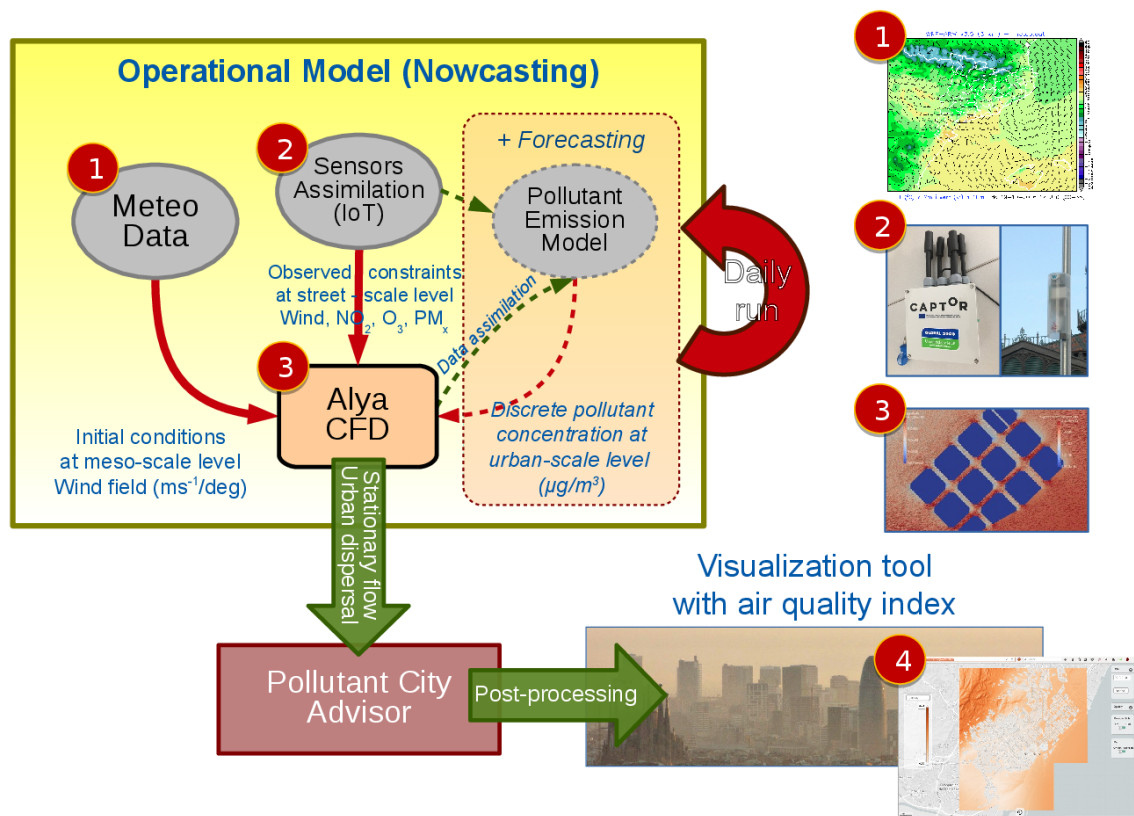


Figure 18: Conceptual sketch of the urban-scale air quality forecast system

On top of this model, the most critical components have been chosen to take advantage of the LEGaTO platform. Actually, within this workflow, the Alya CFD-based simulator is the main compute-intensive component and the one that takes longer to run, and therefore some of its kernels are the main target to be ported to the LEGaTO stack. The source code can be found under <https://github.com/legato-project/FinalSoftwareStack>

This use case aims at demonstrating that monitoring of urban air quality through CFD simulations is feasible for short term forecasts (also known as nowcasting) in an operational workflow built on top the LEGaTO stack.

4.1 Metrics & Optimization Goals

Due to the operational context of the Smart City use case, the air quality model must be constantly executed every update period with new input data from pollution sensors and the meteorological agency. The update period can be so small, between 30 minutes and 1 hour, that it can be critical for running the whole simulation. In this scenario, the LEGaTO stack is pivotal, both in terms of leveraging the processing capabilities to shorten simulation times and improving the energy-efficiency of a highly-demanding urban-scale air quality modelling system. In order to evaluate the LEGaTO implementation, two main metrics must be gathered:

- Use case performance: elapsed time per CFD simulation (*metric 1*).
- Energy-efficiency: Joules per CFD simulation and FLOPs/watt (*metric 2*).

Using the previous two metrics, the following LEGaTO optimization goals will be evaluated for this use case:

- Allow complex urban area simulations (*metric 1*): leverage total execution time objective (10x faster).
- Increase update frequencies in the operational context (*metric 2*): one order of magnitude in energy-efficient objective (10x energy savings).
- Improve FPGA designer productivity: ease the porting of compute-intensive kernels to FPGAs devices using OmpSs model.
- Increase the MTBF factor: support fault tolerance by means of agnostic FPGA task checkpointing, allowing task replication and reduction of failures during simulations (5x increase).

4.2 Baseline Benchmark

Once that the functionality of the LEGaTO version is checked, a baseline test case must be used to benchmark the Smart City use case. This test set will be used to evaluate the optimization goals of the LEGaTO implementation with respect to the original code using the agreed metrics. The test set was run on MareNostrum IV supercomputer, and the most important metrics were obtained as a reference for the baseline. These metrics will be used in future analysis to evaluate the improvement of the LEGaTO implementation.

The baseline evaluation is performed using a pure-MPI Alya CFD version running the Smart City application. Then, we will evaluate the application by applying the LEGaTO optimizations mentioned above. These will include OmpSs taskification to express efficient parallel execution of the application, using single or mixed-precision floating point with minimal loss of quality, and vectorization for efficient execution on accelerators.

The test set used for the baseline comparison is composed of two tests, a minimal mesh and a full-size mesh:

- *Test 1*: proof of concept mesh with a single square building block of 22 meters height and 150 meters long on each side (see Figure 19). The whole urban-area mesh is composed of

almost 1 million tetrahedral elements, with a higher element refinement on the boundary layers of the building and on its leeward wall (downwind side of the building).

- *Test 2*: full-sized mesh of Barcelona city geometry of 500 meters high (see Figure 20). The whole urban-area mesh is composed of 80 million tetrahedral elements, each element with a minimum resolution (length of the element faces) of 15 meters at surface level (streets and building walls).

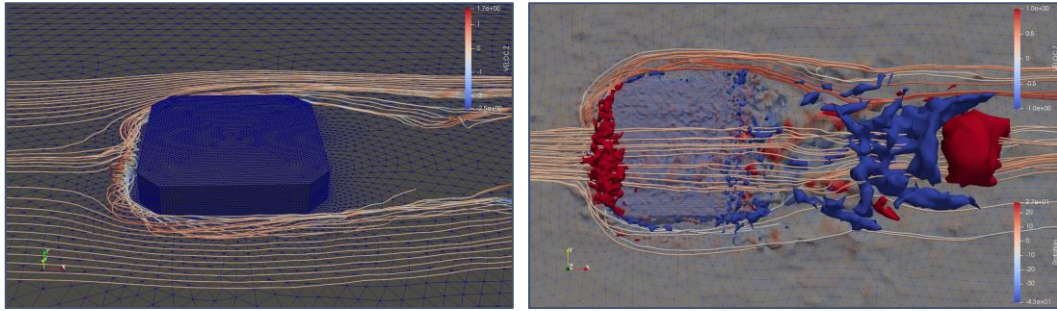


Figure 19: Mesh with a single building block

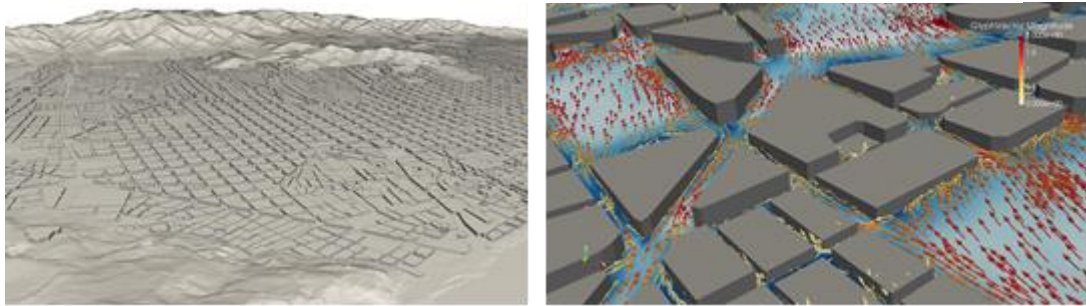


Figure 20: Mesh with a full-sized urban mesh from Barcelona city

As initial condition for the simulations, a west-component (180 degrees) logarithmic wind field profile (10 m/s^{-1} on the top boundary) is used.

Each MareNostrum IV node in our testbed platform is composed of 2 sockets with Intel Xeon 8160 processors with 24 cores each. They run at a frequency of 2.1 GHz and have 96 GB of main memory. One single node was used for *test 1* (48 MPI tasks) and 10 nodes were used for *test 2* (480 MPI tasks). Using this configuration, the Alya's CFD model was run in a pure-MPI parallel way over 100 time-steps for both tests. Different metrics were taken from the job execution for future comparisons with the FPGA versions.

On the other hand, we also run some initial tests on the AXIOM board platform. This is an ARM64 bit architecture with a Xilinx FPGA accelerator [9]. However, given the memory constraints on this platform, only the ARM64 processor was feasible to be used with the single building block test (*test 1*). The results obtained on both platforms for metrics 1 and 2 were:

Baseline metrics	Explicit momentum	SpMV	Whole simulation
Elapsed time	12.63 s	2.08 s	28.75 s

Energy consumption	-	-	9741 Joules (2.7 Wh → ~0.3 Wh in the FPGA)
Performance per Watt	-	-	273 MFLOPs/Watt 92.5 GFLOPs (per node)

Table 2: Metrics for test 1 on MN4 using 48 MPI tasks

Baseline metrics	Explicit momentum	SpMV	Whole simulation
Elapsed time	-	-	29800 s
Energy consumption	-	-	195190 Joules (54.2Wh)
Performance per Watt	-	-	12.96 MFLOPs/Watt 84.92 MFLOPs (per node)

Table 3: Metrics for test 1 on AXIOM board using 4 OpenMP threads

Base line metrics	Explicit momentum	SpMV	Whole simulation
Elapsed time	654.9 s	268.7 s	1505.45 s
Energy consumption	-	-	4733167 Joules (1.3 KWh → ~0.1 KWh in the FPGA)
Performance per Watt	-	-	175 MFLOPs/Watt 55.2 GFLOPs (per node)

Table 4: Metrics for test 2 on MN4 using 480 MPI tasks

4.3 Development Status & Optimization Path

Within the Smart City use case, the CFD-based simulator is the main core of the whole application. To this end, an urban-scale wind forecasting system was developed, based on coupling the meso-scale WRF model with BSC's Alya modelling system. On top of this model, the most compute-intensive kernels are being ported to take advantage of the LEGaTO platform. Two main hotspots have been identified as targets:

- Explicit momentum solver: long numerical code where the submatrix for each element in the mesh is computed and assembled in the global system. This is the main optimization target due to their computational cost (~60% of the global time). (*kernel 1*)
- Sparse Matrix Vector operation (SpMV): Sparse L2 BLAS operation used in both explicit momentum (just once per time-step) and implicit pressure solver (iteratively until

convergence criteria is reached). Due to its relative computational cost (~10-15% of the execution time) this operation has been also considered to be ported. (*kernel 2*)

The porting of those two kernels to the LEGaTO stack requires at least two operations: the full rewrite of the Fortran kernel codes to C language, and the kernel taskification with OpenMP programming model. Note that both algorithms are dominated by the indirect memory accesses due to the unstructured mesh connectivity.

The first step consisted in developing a fully ported of both kernels to C and has been taskified with OpenMP. The kernel has successfully been compiled and run on Intel and ARM64 platforms (Intel Xeon Platinum 8160, Cavium ThunderX2 64-bit ARMv8 [10] and AXIOM board [9]), where the baseline test-case for the Smart City use case was executed to validate the correctness. These kernels were also annotated with OpenMP@FPGA directives in order to run on the Xilinx Zynq Ultrascale+ FPGA on the AXIOM board. However, the existing memory on that device was not sufficient to execute the test-case. Its 4GB of memory were not enough to load the mesh, allocate Alya physical variables, use the OpenMP backend (libnanos), and allocate the required buffers to transfer data back and forth to the FPGA processor. As a workaround and for future testing, the development has been moved to a PCIe Alphasdata FPGA board [11] which incorporates more memory.

The second step consisted in developing an optimal implementation of the SpMV on the FPGA board. The main reason to start with kernel 2 is that improvements developed on this kernel would easily be applied on kernel 1, which is more complex to develop. On the SpMV, the main performance issue to consider is the storage format of the sparse matrix. The sparsity pattern of the matrices arisen from Alya is highly irregular since it represents the connections from the cells of an unstructured mesh. The number of non-zero elements per row depends on the geometrical shape of the cells, and for tetrahedral meshes is 5 for almost all the cells. The rows that represent the connections with boundary conditions have less non-zero elements, zero padding is applied on those rows to introduce regularity. CSR is the most common storage format used on sparse matrices, however in the Smart City use case the optimal storage format is the sliced ELLPACK [12].

Three different implementations using FPGA were compared with the CPU version on the PCIe Alphasdata FPGA board: i) the naïve implementation using the same ELLPACK format than the CPU ii) an optimized CSR found in the literature [13], and finally, iii) a tuned ELLPACK format that aims at reusing the data by implementing a cache like structure that optimizes the reading of multiplying vector.

For testing the implementations, a set of matrices arisen from Alya with sizes ranging from 50,000 rows up to 800,000 rows was utilized. The optimized CSR has the limitation that the resulting vector needs to be fully loaded into local memory, and therefore, the matrix size is limited by the architecture specifications. The execution times are shown in Table 5.

	50,000	100,000	200,000	400,00	800,000
NAÏVE	0.138	0.277	0.545	1.183	2.335
CSR OPTIMIZED	0.112	0.221	0.425	0.903	1.758
ELLPACK CACHED	0.017	0.025	0.036	0.075	0.142
CPU	0.001	0.003	0.005	0.012	0.023

Table 5: Execution time in seconds for the SpMV using different implementations on the FPGA

The results show that the optimized FPGA outperforms in up to 16 and 12 times the naïve implementation and the optimized CSR, respectively. However, in its best case runs 6 times slower than the CPU version. The unstructured memory accesses are difficult to map on the FPGA, needing of extra logic on the die that reduces the possibility of exploiting the parallelism. These results lead to the decision of changing the LEGaTO component to the NVIDIA Xavier boards as explained in the next section.

4.4 Final numerical results

The LEGaTO component utilized for the numerical results consisted in two NVIDIA Xavier boards connected through a PCIe link developed at Bielefeld. One of the characteristics of the Xavier boards is that the maximum energy consumption can be altered by changing the frequency, power budget or the number of cores available. Each board has up to six different setups, so called energy modes. In our case, the most representative modes are:

- Mode 0: Full computing capability, at 2265 MHz frequency, and no power budget (8 cores and GPU available)
- Mode 3: Full computing capability, at 1200 MHz frequency, and a power budget of 30W (8 cores and GPU available)

As a baseline we have considered the execution using one MareNostrum₄ node. The node is composed of two Intel Xeon Platinum 8160 summing up 48 CPU cores.

Alya parallelization strategy is based on a geometrical domain decomposition. A partition of N subdomains requires of launching $N+1$ MPI processes. The process with rank 0 coordinates the work, while for each of the remaining N processes (workers) a unique subdomain is assigned. Point-to-point MPI communications transfer the data between adjacent subdomains, while collective communications involve all the processes. CUDA is used to engage the Xavier board GPU. An overlapping strategy is utilized to hide part of the overhead of performing MPI+CUDA communications. Moreover, all the MPI processes (workers) launch CUDA kernels to the same GPU.

The execution time of the time integration of 1000 steps has been considered. Note that the pre-processing stages are not measure here since in a real simulation its costs become negligible compared with the millions of time integration steps needed. The results are show on Figure 21.

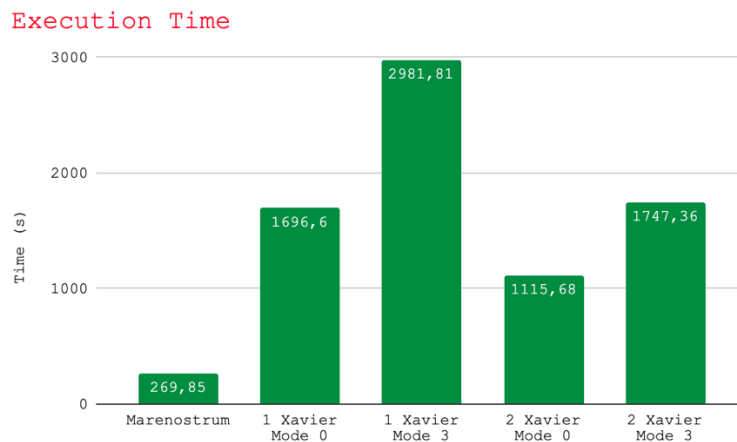


Figure 21: Total execution time of Alya test case for different configurations

The results show that the faster execution utilizing a single Xavier Board is obtained when the energy mode with maximum consumption is activated (mode 0). This is also true when engaging

both boards of the prototype, however the performance doesn't scale linearly. Only a 50% of acceleration is attained by adding an extra Xavier board in the calculation. The performance degradation can be explained by the overhead introduced by the PCIe interconnection between the cards when point-to-point or collective communications are needed.

In addition, a MareNostrum4 node (48 cores) runs 6.3 times faster than a single Xavier board (8 cores+GPU) at its maximum energy mode.

Despite the slower results of the prototypes, a more interesting perspective can be obtained by analysing the power consumption on each platform. In MareNostrum4, jobs are launched using the Slurm Workload Manager [14]. While running, Slurm collects performance and power consumption information that we gather and analyse. For the Xavier prototypes, the tegrastats tool [15] provided by NVIDIA reported such information. In both cases, the full power consumption of the node was considered. The results presented in FLOP per watt are depicted in Figure 22.

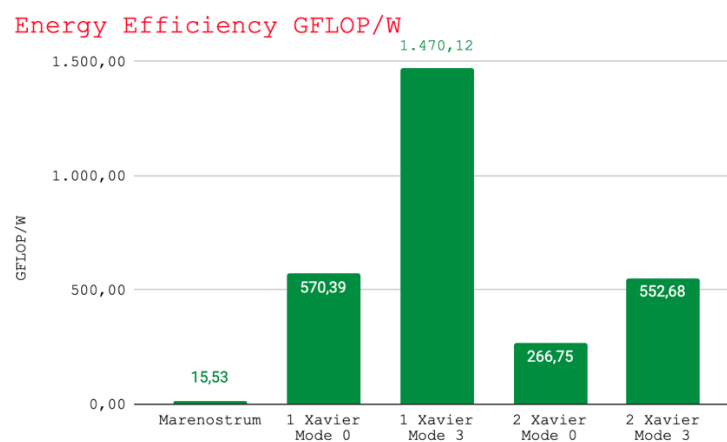


Figure 22: Energy efficiency in GFLOP/W for different setups

The energy measurements show that one Xavier board in mode 0 is 35 times more energy efficient than the Marenostrum4 node. Moreover, if the Xavier mode is utilized in mode 3, the energy efficiency achieved is 95 times at the cost of 11x slowdown. When using 2 Xavier boards, the speedup obtained is only 1.7x (due to the overhead of communications), and as a result, the performance and energy - efficiency do not get benefits. One may consider to use the 2 Xavier boards in mode 0, to get better performance at the expense of being less energy efficient

The current configuration of the Bielefeld Xavier prototype allows engaging only up to two Xavier nodes. A rough estimation of the performance using more nodes is possible by measuring solver calculations and communications times separately for different workloads.

The time of calculations are measured locally using one Xavier node for matrices ranging from 50,000 to 1,600,000 rows. Note that by doing this we are assuming an optimal distribution of workload among the nodes. The next step consists in estimating the communication costs only using 2 nodes. For this task, we have measured the communication between two nodes using different message sizes. The main assumption is that the slowdown in performance when passing from 1 to 2 nodes is going to be just scaled by the message size when passing from 2 to 4 nodes. The calculation and communications times are shown in Table 6.

Nodes	1	2	4	8	16	32
Local workload	1,600,000	800,000	400,000	200,000	100,000	50,000
Calculations	5.64	2.18	1.07	0.42	0.29	0.16
Communications	0.03	0.18	0.78	1.03	4.03	12.66
Total Time	5.67	2.35	1.85	1.45	4.32	12.83

Table 6: Execution time in seconds of the calculations and communications of Alya's solver for different workloads

The final scalability is obtained by adding the calculation and communications costs. The estimation of the scalability of the solver using up to 32 nodes is depicted in blue on Figure 23. It is clear that the communication cost becomes a bottleneck when the workload per node decreases.

A more optimistic approximation can be done by assuming that the network would behave as in Marenostrum4. For that matter, the communication using two nodes was measured, and then, scaled according to numerical results obtained on Marenostrum4. The dotted on the figure represents the MareNostrum4-based estimation. Note that a better network would improve the performance on the prototypes, and therefore, it would increase the gap in terms of energy efficiency between both configurations.

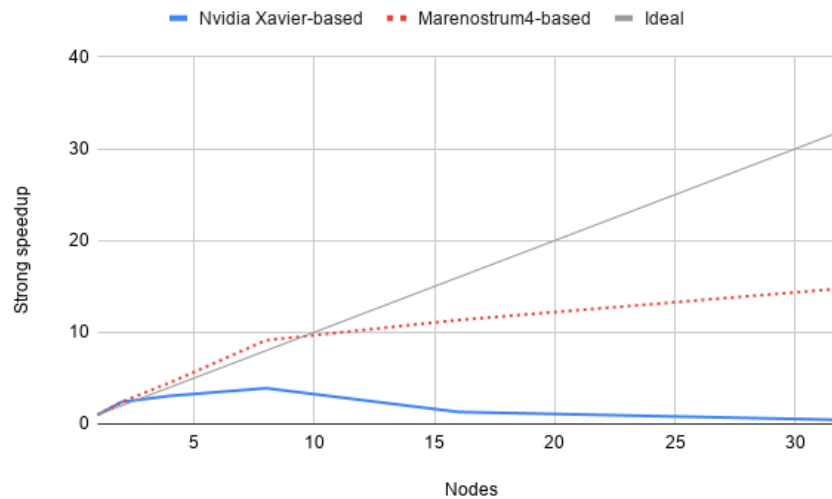


Figure 23: Estimation of the scalability of Alya

4.5 A low energy-consumption Alya-oriented cluster

Alya is one of the twelve simulation codes of the Unified European Applications Benchmark Suite (UEABS), and therefore takes part in many scientific projects that result in using a large part of the PRACE infrastructure. On Marenostrum4, Alya-based simulations are one of the main applications that exploit the supercomputer. Over the last year, the majority (~60%) of Alya-based simulations engaged an average of 50 computing nodes (2,400 CPU-cores) on MareNostrum4. Consequently, it would make sense to create an infrastructure oriented to solve simulations on that size range, since it would have the largest impact on Marenostrum4 power consumption.

An estimation of the size of the new infrastructure can be obtained considering the optimistic scalability estimations. For this purpose, three assumptions are needed:

1. the simulation results on the new infrastructure should be attained on a similar time frame
2. a node of Marenostrum4 is 6.3 times faster than a Nvidia Xavier node (see Figure 20)
3. up to 9.3 times of acceleration would be obtained by using 8 Xavier nodes (see Figure 22)

The conclusion is that the new low-energy consumption cluster should be composed of 400 Nvidia Xavier nodes that would deliver the same execution time on the average usage of Alya, than the usual 50 Marenostrum4 nodes.

The ratio of power consumption between a node of Marenostrum4 and a Xavier node is approximately 35 times when running an Alya use case. Then, the new cluster that would need 8 times more nodes would consume approximately 4.4 times less energy than Marenostrum4. Since the new cluster would be used on 60% of Alya executions, the other 40% would still consume the same. Therefore, the overall power consumption of Alya would be reduced by 47%.

4.6 Increase robustness for Alya at large scale

The smart city use case sometimes requires runs with very high resolution and long timelines. It is not surprising for some of those runs to take hundreds of hours of execution, spanning over several weeks in multiple runs of 24 to 48 hours. In addition, these simulations usually run on thousands of processes, consuming hundreds of thousands of CPU hours.

Supercomputers at the petaflop scale with thousands of computing nodes, such as the Marenostrum4 supercomputer at the BSC, usually observe around three node failures per day, translating into a mean time between failures (MTBF) of about 6 to 8 hours. At this scale, any failure immediately translates into huge energy waste. Therefore, we apply checkpoint/restart (see chapters 5.1 and 5.2 of Deliverable 3.3 [16]), to substantially decrease the amount of re-computation required upon a failure. However, checkpointing had not been applied to the GPU version of Alya previously. It is not easy to checkpointing heterogeneous applications as one must work with multiple memory devices, the host and the device memory.

In LEGaTO, we have created a tool capable of checkpointing heterogeneous applications running on GPU clusters, such as Alya. Moreover, our tool leverages multiple storage systems implementing multilevel checkpointing, and we parallelize device-to-host data streams, with writing into reliable storage (i.e., SSDs, PFS). In this way, we have not only enabled fault tolerance for heterogeneous applications but also decrease the checkpoint and recovery time by up to 15.23X and 5.21X, respectively. Furthermore, for the same amount of waste, i.e., 15%, our checkpointing technique can sustain execution in systems with 6.3 times smaller MTBF [17].

5 Infection Research

Recently, biomarkers are used widely in medical research to detect a disease early, for diagnosis, in making prognostic or risk assessments, which nowadays plays an important role in modern clinical and preventive medicine. By modern technologies like microarrays, next generation sequencing, and mass spectrometry, researchers can measure many biomarkers that may exceed ten thousands [18] [19] [20]. With this development comes the challenge to find biomarkers to detect the risk to have a disease or diagnose it with high confidence.

To avoid wasting time and money, researchers do pilot studies with a small number of observations as necessary first step for biomarker discovery. Because of the small number of cases and the large number of biomarker candidates, any result might be caused by random effects and statistical significance cannot be proven. For that, researchers try to reduce the number of biomarkers and extract the most informative ones from these pilot studies to then increase the sample size and achieve an adequate statistical power.

5.1 Description of the accelerated biomarker discovery workflow

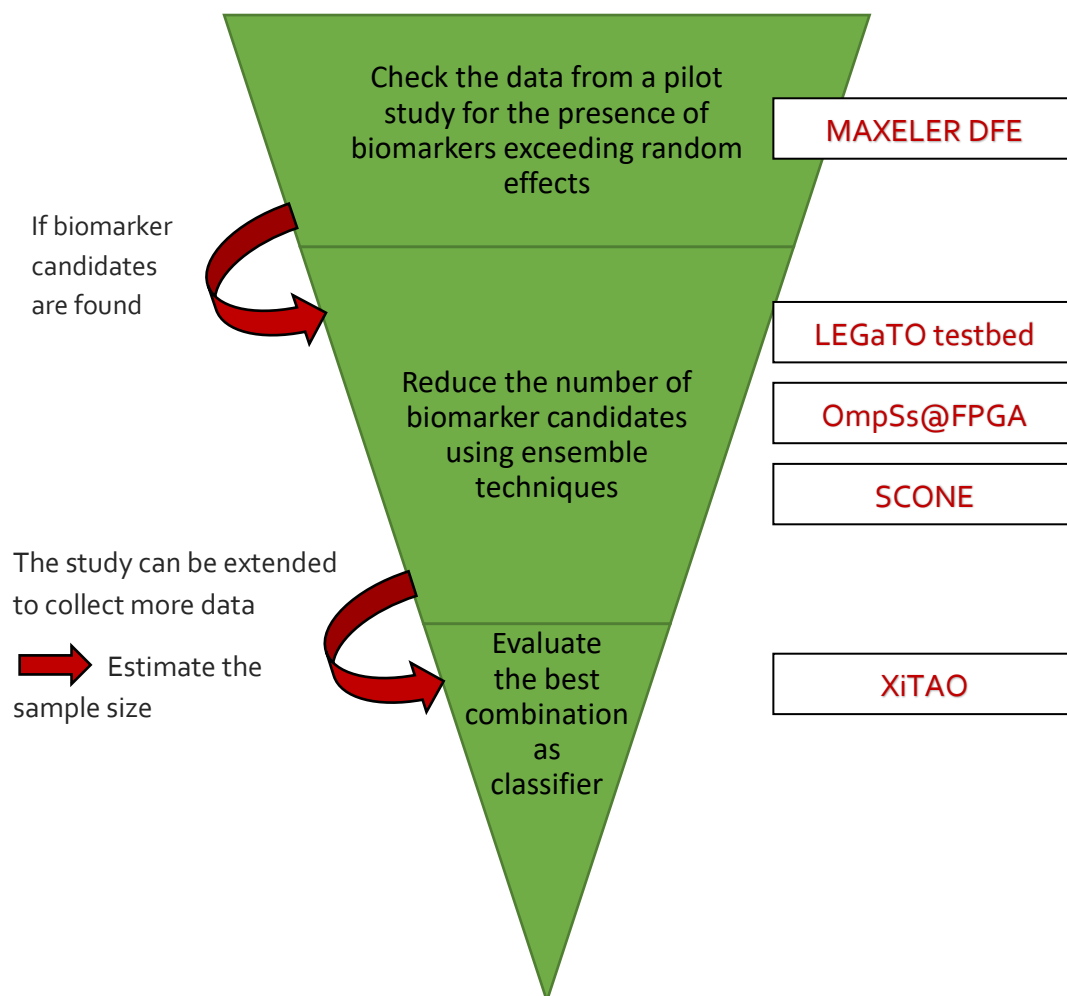


Figure 24: Accelerated biomarker discovery workflow

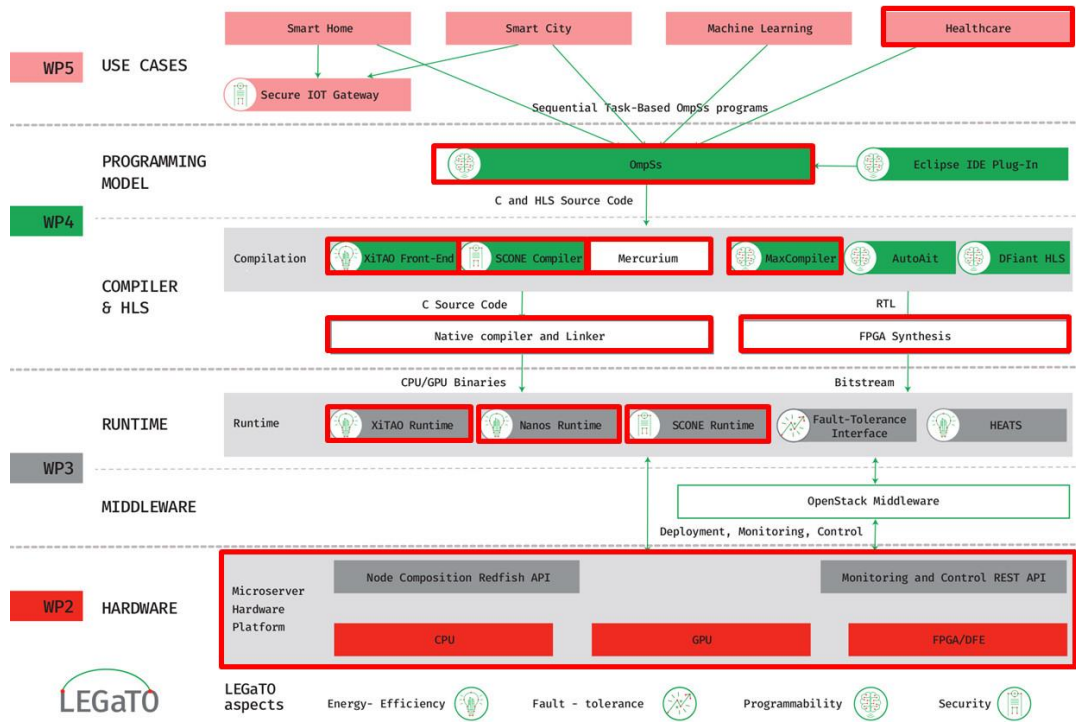


Figure 25: Healthcare Toolchain

The discovery of biomarkers, from the pilot study to the biomarker that can be used in practice, involves several different steps of data analysis, as shown in Figure 24. In order to accelerate the workflow described in detail below, we have used several components of the LEGaTO toolchain, see Figure 25.

In a first step, we check the data for the presence of biomarkers that have an association with an outcome exceeding the pure random effect. Our new HiPerMab curve method (to be published soon) shows how many biomarker candidates in the real data set exceed a specific value of entropy (a performance measurement) compared to a random data set and a confidence interval, see Figure 26. To measure the probability of how many biomarkers exceed a specific value of entropy, simulations have to be done.

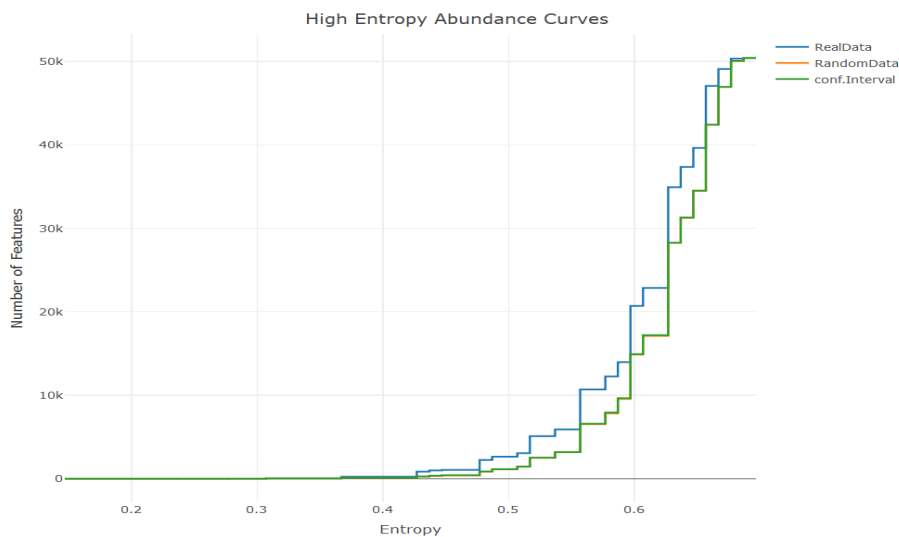


Figure 26: The number of biomarkers with a performance value that does not surpass a specific value of entropy

In cooperation with Maxeler and their MaxCompiler, the computing time for the simulations has been reduced dramatically by accelerating key parts of the computation with Maxeler DFEs. As a first step, the application was profiled for its most computationally intensive components. These ones naturally represent the best opportunity for application speed-up. Due to the close link between efficient processing in terms of speed and energy on DFEs, successful application speed-up by offloading computation-intensive parts to DFEs also typically leads to significant energy reductions. In this case, the cut index function which is part of the R package "discretization" was chosen for DFE offloading and acceleration. This process meant the cut index functionality was re-implemented in Maxeler's MaxJ language and compiled to a state-of-the-art MAX5 DFE card with MaxCompiler. In order to facilitate the integration of the DFE accelerator functionality into the original R application, a C++ interface layer was created in order to call the Maxeler SliC API [21].

To reduce the number of biomarkers (also called features), we extract the most informative ones simply by ranking them according to AUC, entropy, sensitivity, specificity or other metrics. This approach of feature selection by filtering is relatively fast but does not take relationships between biomarkers and biomarker combinations into account.

Along with the advancement of high-throughput technologies that allow measuring the whole or large parts of the genome, transcriptome, proteome or metabolome, came a strong hope to find a single biomarker for each disease or state of a disease to be diagnosed with very high certainty. However, this dream did not come true, and it seems to be unrealistic from today's point of view. Biological systems are probably too complex for simple single-cause single-effect associations. Nevertheless, there are biomarker candidates that show a high correlation with specific diseases but are not reliable enough to function as predictors for the presence of a specific disease alone. Instead of relying on a single biomarker, the idea is to combine biomarkers that are not good enough for the diagnosis of a specific alone but can jointly provide a diagnosis with high certainty. Therefore, we additionally apply embedded feature selection with tree-based methods (random forests and gradient boosted decision trees), including the calculation of the robustness of the feature selection. Building thousands of models and finding the best set of hyperparameters is computationally expensive. This step of the workflow is accelerated based on the results of the accelerated HiPerMab curve method from the first step. In addition, we use a fast C++ implementation with Python bindings (lightGBM [22]) and a hyperparameter optimization software (optuna [23]), which generally speeds up the selection of hyperparameters. This optimization was developed on the LEGaTO testbed in Bielefeld and was made possible by the technical support provided there.

To verify results that are not very robust, we want to use ensemble techniques. That is why we have added another new approach to the workflow. In this further algorithm, we do not predict the class directly from the training data. Instead, we predict every biomarker candidate by LASSO regression [24]. From the importance of the included class label for the prediction and the regression error, we calculate a weight for this specific biomarker candidate. We assume that the higher the weight, the more relevant the biomarker candidate is. Due to timing reasons explained in the deliverable 5.2 and to use more diverse algorithms we switched the regression method from lightGBM ([15] a tree-based method) to LASSO regression [17]. The LASSO regression has only a single hyperparameter that can be determined quickly. In addition, the LASSO algorithm is less complex and faster. As a pre-processing step for the LASSO regression, the skewed biological data must be scaled and transformed into a distribution close to a normal distribution. An example with randomly generated artificial data is shown in Figure 27. For this necessary transformation, we use the Yeo Johnson Transformation [25].

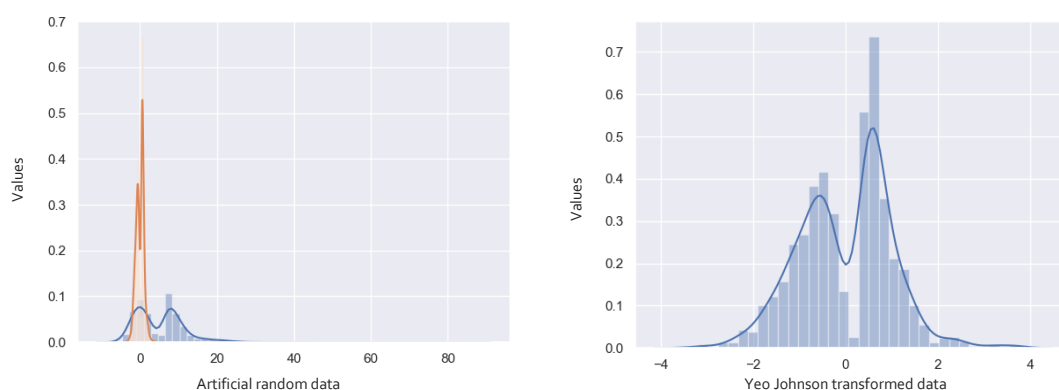


Figure 27: Skewed random data versus the transformed data

To accelerate this time-consuming transformation for large data sets we use OmpSs@FPGA. The implementation of the transformation was done in C and subsequently modified with corresponding OmpSs@FPGA and HLS instructions so that the time-consuming part of the program could be processed in the Programmable Logic of Zynq-7020 ARM/FPGA SoC Development Board. The transformation of an input vector was processed in a hardware pipeline so that the next iteration starts before the previous iteration is finished.

At no point the use of a hardware description language (HDL) or knowledge about it was necessary, so OmpSs@FPGA fulfils the approach of a high-level synthesis very well and is a proof for increased designed productivity compared to HDL languages. Another advantage is the common address space of the Programmable Logic (PL) and the Processing System (PS). This allows a realization of the task like on a normal PC.

The BSC provides a boot medium for the supported systems, on which operating system, runtime environment and libraries are installed. The BSC relies on the Docker technology for the distribution of the software for the OmpSs@FPGA project. By providing the appropriate image, the developer can download the desired OmpSs version for the target system as an image and start it in a container. In this container, an operating system with the necessary toolchain is available and can be used immediately. Only the "FPGA specific vendor tools" have to be inserted into the path of the started container. In the case of the Zedboard, which has a Z-7020 from Xilinx as FPGA, it was the Vivado Design Suite from Xilinx.

This type of acceleration could also be applied in other similar applications that transform data for biomarker discovery and be added to the biomarker discovery workflow.

Furthermore, we integrated the part using the LASSO regression with SCONE to test and show the possibility of secure data processing even in the cloud. Sensitive medical data can be handled according to data security aspects.

Combining the information of the different approaches explained above, we select a subset of biomarker candidates. We then estimate the sample size for extended studies in order to achieve adequate statistical power. After collecting enough data, we repeat the biomarker selection process and evaluate the best combination of biomarkers using Random Forests.

In our previous version of this algorithm, we selected the 18 most promising biomarker candidates from a data set with 66 samples and calculated every possible combination within this selection. From a statistical point of view, however, the number of biomarkers in the classifier should not exceed 10% of the sample size, and from a medical point of view, researchers prefer to have as few biomarkers in the model as possible. For this purpose, we optimized the code in a first step by

reducing the number of biomarker candidates contained in the classifier to a maximum of 6. We have parallelized the adapted code for the new baseline measurements. In order to really achieve a significant acceleration, we have implemented the complete algorithm again in C++ and accelerated this implementation with XiTAO (c.f. section 8.1). We chose XiTAO because it is very simple to integrate into C++ code and the program can be run on any computer. Depending on the number of available cores, the acceleration can easily be adapted to the respective hardware environment. This part of the software can be reused without any special hardware requirements.

The optimised software used in the Infection Research use case can be requested via Prof. Dr. Frank Klawonn <frank.klawonn@helmholtz-hzi.de>, who is leading this use case at the Helmholtz Centre for Infection Research.

5.2 Metrics & Optimization Goals

We could reach our main optimization goal to achieve a significant speedup in several parts of our biomarker discovery workflow. This enables us now to handle larger data sets (especially in terms of biomarker candidates). The number of biomarkers and the size of the data sets we are able to handle is crucial for our future projects. We are part of the project consortium i.Vacc (paving the way towards individualized vaccination) [26]. The project started in 2020 funded with one million Euro within the framework “Big data in the future life sciences” of the Lower Saxony Ministry for Science and Culture in Germany [27]. In contrast to our other projects where we usually handle data from one so-called omics fields (metabolomics, proteomics, transcriptomics, genomics), we will look jointly at different types of omics, increasing the number of biomarker candidates drastically. Analysing data from three or four different omics fields jointly does not mean that we have to apply our algorithms three or four times but the size of the biomarker candidate data set increases by factor of three or four. Our goal is therefore to work with data sets at least three times as large as the ones we could handle before. This means we want to extend the number of biomarker candidates to >50,000 and the sample size to >100. With the optimizations achieved in the LEGaTO project, these calculations are now possible.

We analysed data with more than 50,000 biomarker candidates and estimated a probability of less than 1/50000, which means that we performed 5 million simulations (with a 95% probability that the (relative) error was less than +/- 20 %). Also, we can apply our feature selection methods to data sets with more than 50,000 biomarker candidates and achieve better results by being able to calculate more trials in less time saving a significant amount of energy. Furthermore, we are now able to find biomarker combinations from biomarker sets that are larger than 20.

5.3 Benchmarks

To evaluate our progress, we measured the runtime and the energy consumption of our algorithms.

The data set used for our benchmarks belongs to a pilot study which used whole-genome microarray analysis to investigate the transcriptomes of periprosthetic hip tissues to identify genes that are differentially transcribed between chronic periprosthetic hip infection and aseptic hip prosthesis loosening. Differentiating between periprosthetic hip infection and aseptic hip prosthesis loosening can be challenging, especially in patients with chronic infections. 24 samples are represented in the rows. 12 cases with chronic periprosthetic hip infection, which is represented as infected and 12 cases with aseptic hip prosthesis loosening, which is represented as uninfected. The data has 50416

genes (biomarker candidates) each one is represented in one column. We analysed this data set with more than 50,000 biomarker candidates and performed 5 million simulations.

To check the data from this pilot study for the presence of biomarkers exceeding random effects with HiPerMab curves we accelerated the most time and therefore also energy-consuming part of the code. For the benchmark below, we considered the cut index function.

As energy measurements in Jülich have not been possible because of a software bug, the energy measurements for the simulations have been carried out on the Bielefeld testbed using the internal power measurement possibilities of the RECS|Box system as described in D2.1 section 6.3.

In LEGaTO there are two MAX5 testbeds:

1. Juman DFE system at the Juelich Supercomputing Centre and
2. RECS|Box equipped with a MAX5 card located at Bielefeld.

The calculations for the standard R version have been carried out on a Microserver with an Intel Xeon E3-1505M v6 CPU with 4 cores and 32 GB RAM.

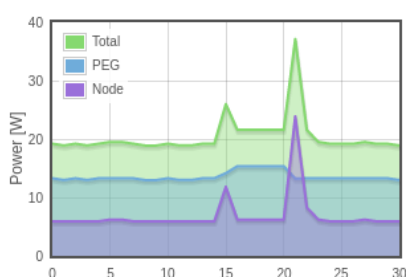


Figure 28: Energy consumption of the Microserver and Maxeler MAX5 card

The energy consumption of the optimized version is shown in Figure 28. The blue part (PEG) shows the energy consumption of the MAX5 card, the violet part (Node) monitors the CPU and the green part shows the overall energy consumption of the complete node 4 of the testbed in Bielefeld (Total).

In Table 7, the execution time and the energy consumption for 5 million simulations is listed for the execution of the standard R package parallelized and running with 7 threads and the optimized version using one MAX5 DFE card. MAX5 DFEs use a Xilinx Ultrascale+ VU9P

and the detailed architecture of the card has been previously described in D2.1.

5 million simulations	Standard R	Maxeler DFE
Elapsed time (s)	13990.0	5.588
Speed-up	1 (baseline)	2503.6 x
Energy consumption (J)	906552.0	117.6
Factor of saved energy	1 (baseline)	7708.8 x

Table 7: Measurements for 5 million simulations

The second acceleration is a software optimization based on the results of the HiPerMab curves and the best result of the ranked biomarker candidates. This optimization was developed on the LEGaTO testbed in Bielefeld and made possible by using the hardware provided there.

For this benchmark we measured the duration and energy consumption in the testbed in Poznan using the node xeon-d-02, equipped with an Intel Xeon D-1577 CPU with 16 cores and 32 GB RAM. Table 8 below shows a comparison of the elapsed time and energy consumption of the first 97 trials of the hyperparameter optimization for the pilot study data set described above including 50416 biomarker candidates. For one trial of the hyperparameter optimization for this specific data set up to 258 steps from 264 can be saved, which is an improvement of 97.7%. These savings are possible by incorporating the information obtained from the accelerated HiPerMap curves calculated in the

previous step. With the optimization 24975 steps of the nested leave one out cross validations for 97 trials in this benchmark could be saved. And thus, the duration and energy consumption can be reduced. The improvements in duration and energy consumption are calculated based on the number of steps saved, and we assume that calculating fewer steps will require proportionally less energy and less time. Due to random parts of the algorithm, it is not possible to take exact measurements for a direct comparison. But the number of steps is a clear metric to compare both versions for this example.

Embedded Biomarker subset selection	Standard Version	Optimized Version	Saved
Overall iteration steps of all 97 trials	25608	633	24975 iterations
Duration	72 hours	~1 hour 48 minutes	~70 hours 12 minutes
Speed-up	1 (baseline)	40x	97.53% of the time
Energy consumption	2232 Wh = 8035200 J	~55 Wh = 198000 J	~2177 Wh = 7837200 J

Table 8: Measurements for the first 97 trials of the hyperparameter optimization of the Embedded Biomarker subset selection

The Yeo Johnson transformation is needed for the pre-processing of specific machine learning methods. For the biomarker discovery workflow, it is used as pre-processing step for the LASSO regression. The toolchain OmpSs@FPGA allows to implement the Yeo-Johnson transformation on a supported system on chip (the ZedBoard used for a proof of concept) and to accelerate it in the PL afterwards.

By measuring the energy consumption of the Zynq-7020 ARM/FPGA SoC Development Board it should be checked if the increase in speed is not compensated by a correspondingly higher energy consumption. Therefore, the energy consumption during program execution was determined. Using a current probe (Tektronix TCP0030A), the current is measured at the positive pole of the power supply connected to the ZedBoard while the input voltage is kept constant at 12V DC. The current changes with the power consumption required during program execution. For Direct Current (DC) probes, the current is evaluated with a Hall sensor, the value of the Hall voltage is proportional to the required current. The used current probe can be connected directly to an oscilloscope. A Tektronix device (type MDO4054B-6) was used.

The runtime in seconds is given as an average value rounded to milliseconds over all columns of the input file, at an FPGA clock frequency of 100 MHz.

$$\frac{10.946 \text{ s}}{1.156 \text{ s}} \cdot \frac{331 \text{ mA}}{375 \text{ mA}} \cdot \frac{12 \text{ V}}{12 \text{ V}} = \frac{10.946 \text{ s}}{1.156 \text{ s}} \cdot \frac{3.97 \text{ W}}{4.50 \text{ W}} = \frac{43.48 \text{ J}}{5.20 \text{ J}} = 8.36$$

Yeo Johnson Transformation	ARM processor on the ZedBoard	Z-7020 from Xilinx on the Zedboard
Elapsed time (s)	10.946	1.156
Speed-up	1 (baseline)	~10 x
Energy consumption (J)	43.48	5.20
Factor of saved energy	1 (baseline)	8.36 x

Table 9: Measurements for the proof of concept of the Yeo Johnson Transformation

The data we use in the benchmarks below is linked to current biomarker research for bacterial meningitis. Bacterial meningitis is characterized by a high degree of neuronal damage and a high risk of long-term sequelae [28]. Therefore, one reason for molecular profiling of patient cerebrospinal fluid (CSF) samples is to improve the understanding of pathophysiological networks and mechanisms and to identify disease-specific pathways that could serve as targets for host-directed treatments to reduce end-organ damage and, thus, improve clinical outcome. This data set for our benchmarks contains concentrations of 112 metabolites in cerebrospinal fluid samples from patients with bacterial meningitis (n=32), viral meningitis/encephalitis (n=34, due to herpes simplex viruses, varicella-zoster virus, and enteroviruses), and non-inflamed controls (n= 66).

With this dataset, the benchmarks for the step of the workflow using the LASSO regression integrated with SCONE are executed. To protect patient data and analysis code (Python) we use of SCONE to encrypt data and code and run the computation inside Intel SGX enclaves. To measure the overhead of running our application with Intel SGX, we conduct experiments using a server with SGXv1 support running Ubuntu Linux with a 4.4.0 Linux kernel, equipped with an Intel Xeon E3-1280 v6 CPU at 3.90GHz with 32 KB L1, 256 KB L2, and 8 MB L3 caches, and 64 GB main memory. We evaluate the performance of our application with three modes:

1. native, without SCONE,
2. SCONE hardware mode (HW) which runs with SCONE and activated TEE hardware and
3. SCONE simulation mode (SIM) which runs a simulation without Intel SGX hardware activated.

We report the average evaluation results of 10 runs in Table 10. Running the application with SCONE in HW mode is roughly 1.5x slower compared to the native version due to the fact that the analysis process requires memory-intensive computations and the Intel SGX enclave EPC size is limited to 94 MB. However, running the application with SCONE in SIM mode has very comparative performance compared to the native version. This means with new hardware from Intel in the future, the performance of our application with the security guarantees, is expected to be similar to the native version.

Measuring of code security is not a trivial task. Potential ways are to compare the size of the Trusted Computing Base (TCB), lines of code (LoC) or the binary size of applications; or we can count the known-vulnerabilities of the program. In the SCONE toolchain musl libc [29] is used which has ~60k LoC, however other frameworks use glibc which has ~460k LoC. For this, we are ~7.7x more secure than other frameworks. However, this provides just a very roughly estimation for security.

	Mean	Standard Deviation
Native	32.94 s	0.88
SCONE (HW mode)	49.29 s	1.39
SCONE (SIM mode)	31.44 s	0.94

Table 10: Measurement of the overhead of running our application with SCONE

In order to finally find a classifier that can be used in practice, the classifier performance of small combinations of biomarker candidates is calculated and compared. The following benchmarks were also performed on the meningitis-related data described above. For this part of the workflow, the energy measurements were also carried out on the Bielefeld testbed using the internal power measurement possibilities of the RECS|Box system. In the following Table 11, the execution time and the energy consumption are listed for the calculation of all possible combinations with 5

elements out of the 20 most promising biomarker candidates. All measurements were carried out on the same microserver with an Intel E-2176M CPU with 6 cores and 32 GB RAM.

The Python and XiTAO versions both ran in parallel with 12 threads. Identical hyperparameters and the same algorithm were used in each case. The C++ version is the same code as the XiTAO version, but with XiTAO disabled. The C++ version without XiTAO therefore runs serially. 15504 combinations were calculated.

Classifier evaluation (combinations) C(20,5), 66 samples	Python	C++ (serial)	XiTAO
Elapsed time (s)	104992 s	906 s	193 s
Speed-up	1 (baseline)	215 x	544 x
Energy consumption (J)	4727640	35334	11387
Factor of saved energy	1 (baseline)	134 x	415 x

Table 11: Measurements for 15504 Combinations

6 Machine Learning

Although there are many success stories of deep learning (DL), which has spurred a wave of public and corporate interest in artificial intelligence (AI) and machine learning (ML), there are still many unsolved issues. One of these is how to make the DL models more efficient; both from an energy perspective as well as a cost perspective. This is especially true for embedded applications, e.g. autonomous driving, IoT and robotics. To solve this problem, we are developing a new kind of DL optimisation tool (EmbeDL) that will make DL models use significantly less energy and require less computations in order to be deployed to cheaper hardware. We will demonstrate the technology developed on vision applications relevant for autonomous driving: image classification, object detection and pixelwise semantic segmentation. The technology has been spun off into a startup called EmbeDL. For more information about the company visit <https://embedl.ai>.

6.1 Metrics & Optimization Goals

Multiple of the metrics in the LEGaTO Project are of interest for this use case. As mentioned, the energy consumption of today's state-of-the-art perception systems are too high and need to be decreased. Thus, energy is the most important metric for the ML use case. The second most important metric is to make the models more computationally efficient. By doing so, the same original model can be deployed to cheaper hardware. Latency is also very important, since lower latency can be a critical metric in order to avoid accidents. Increased MTBF and code base security is of course also important for having operational and secure autonomous vehicles in the future. However, practically measuring MTBF and code base security in a meaningful way is challenging. We therefore chose not to focus on these goals, but rather focus on making the use case energy and cost-efficient. Thus, in order of importance, the metrics and goals are:

- 10x energy efficiency (LEGaTO Objective #1)
- 10x faster execution/latency (LEGaTO Objective #6)
- Reduce Cost (LEGaTO Objective #5 & #7)
- Increase designer productivity (LEGaTO Objective #4)

6.2 Optimisation

To reach the goal of accelerating deep learning inference in heterogeneous hardware, a Deep Learning Optimisation Engine, EmbeDL, was developed in LEGaTO.

EmbeDL's technology sits between a front-end interfacing with the major open source DL frameworks and a back end provided by the LEGaTO project as well as tools from various hardware vendors. The DL frameworks supported include all the popular frameworks such as TensorFlow [30], PyTorch [31] or Caffe [32] and open standards for interoperability such as ONNX [33]. The backend supports different hardware technologies including CPUs, GPUs, ASICs and FPGA (and in the future, new emerging chip technologies driven by AI applications).

1. **Easy-to-use Python API:** EmbeDL has an easy-to-use API that makes it easy for data scientists to use EmbeDL's optimization engine.
2. **Graph Analysis:** EmbeDL's technology converts the model specification from any of the front-end specifications (TensorFlow, PyTorch, Caffe or ONNX) to EmbeDL's proprietary

computational graph-based intermediate representation. This is done so that (a) there is one common representation facing the front-end, (b) to allow the extensive surgery on the model performed by EmbeDL's optimization engine and (c) to allow for a common interface towards hardware backends.

3. **Deep Learning optimization engine:** This is the core of EmbeDL's technology. EmbeDL uses many different state-of-the-art model compression techniques to reduce the size (i.e., number of parameters) and computations (i.e., Floating Point Operations, FLOPS) of deep networks without sacrificing too much of accuracy (as required by the user).
4. **Hardware interface and compiler:** EmbeDL is not trying to reinvent the wheel when it comes to hardware support, but rather complements the existing ecosystem. This is especially true when it comes to interfacing hardware, where we use the APIs and tools provided by the hardware vendors that are optimized for their proprietary hardware. However, most importantly, the hardware is abstracted away from the user, which clearly demonstrates the "write once run everywhere" programming promise of the LEGaTO project.

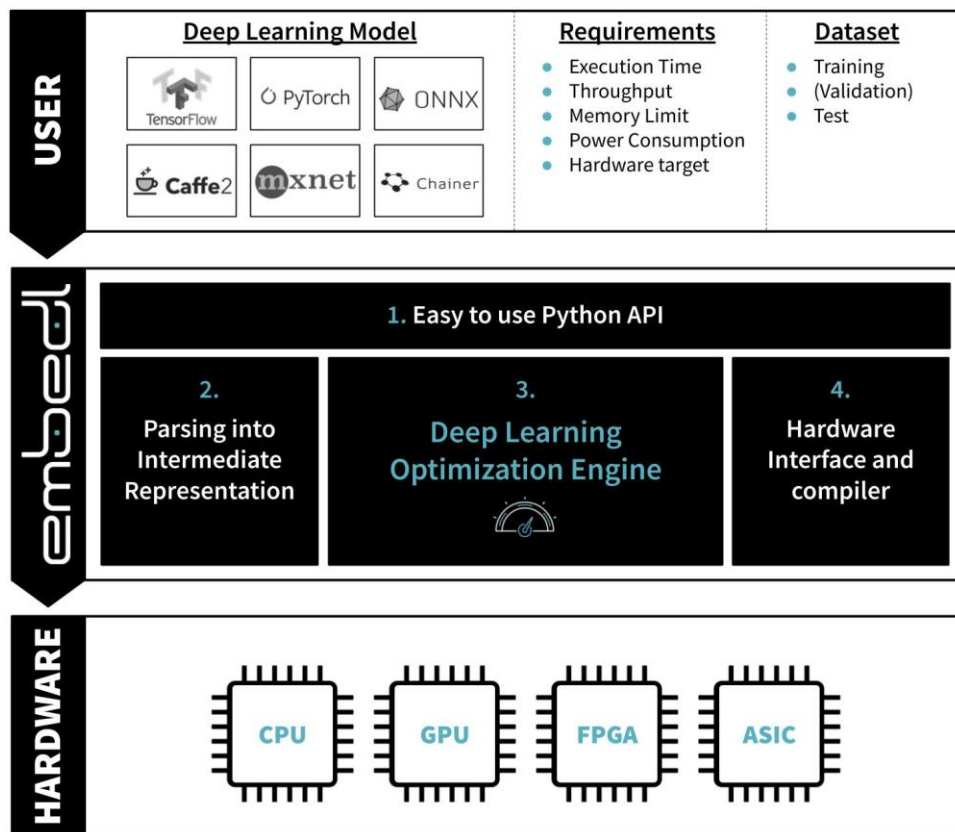


Figure 29: EmbeDL toolchain and ecosystem

6.3 Integration

In this project it is of interest to explore how to port deep neural networks on FPGA devices. This is not something that we had prior knowledge of how to do, and thus this use case utilises the OmpSs in order to target the FPGA in the benchmarks. OmpSs drastically lowers the threshold for a first FPGA implementation and also decreases the time required to test new implementations/methods on FPGA. During the project, we had access to the Microserver Hardware Platform, but in the end, the experiments were conducted in-house for increased controllability, software support and

meaningful comparison with local hardware. The usage of the different parts in the toolchain is presented in Figure 30.

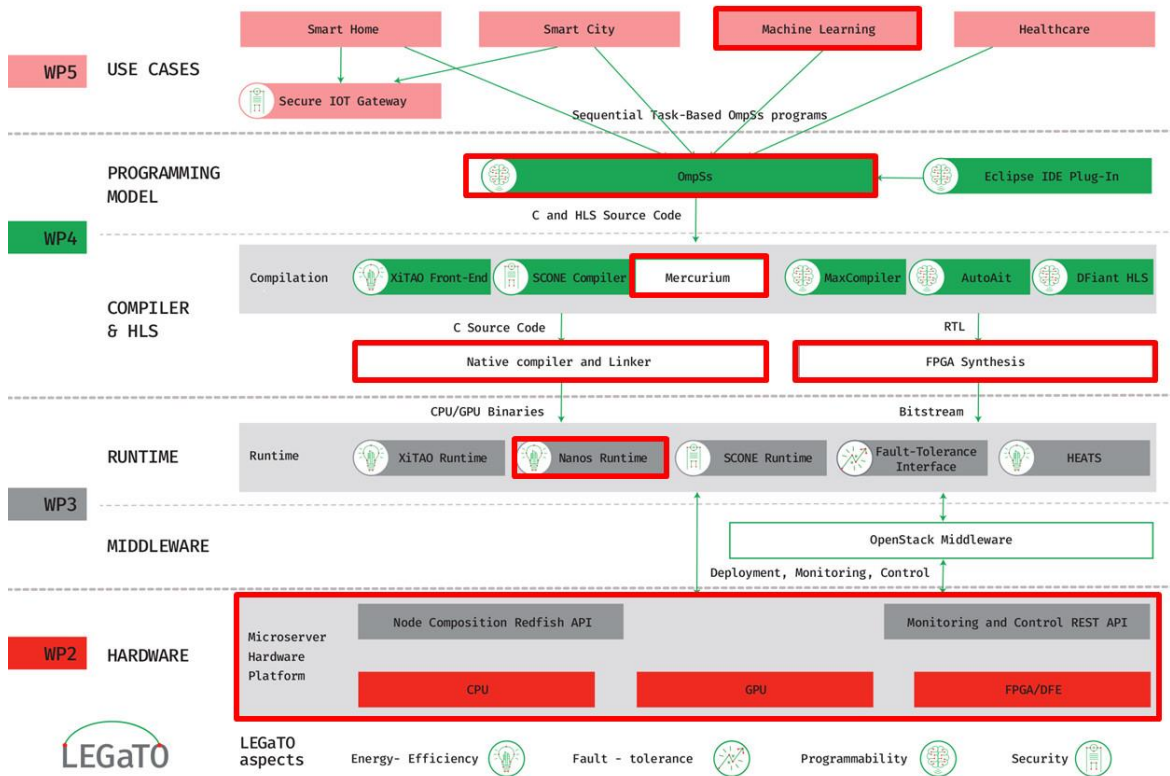


Figure 30: Machine Learning Use Case Toolchain

6.4 Experiments Setup

In the benchmarks, we measure both inferences (frames) passed through the neural network per second (FPS) as our metric for execution time. During our benchmarks, each network works on a single frame at a time, effectively encapsulating latency in the same metric. Furthermore, we use inferences per joule to quantify the energy efficiency of a network. We also present the speedup factor and energy reduction factor between the unoptimised and optimised models.

The inference speed benchmarks on Intel hardware were obtained using “OpenVINO” benchmark application in “sync” mode with “device” option set to “CPU” for Intel Core i9 and “Myriad” for INCS2. All of the CPU cores were running simultaneously. On NVIDIA Xavier (both with and without DLA) and NVIDIA Jetson TX2, speed data was extracted using “trtexec” tool from tensorrt. Both Raspberry PI and Zedboard were measured using time libraries for Python and C respectively. All of the speed benchmarks were obtained by averaging over 100 iterations.

In energy efficiency benchmarks, power consumption and execution time of certain numbers of predictions were measured to calculate energy and subsequently the energy efficiency parameter Predictions/Joules. Time was obtained using the same tools that were used for speed benchmarks. As for power, each hardware platform needed a different technique for power measurements. However, we were able to extract the dynamic power from the total power consumption and the idle power on all of the platforms. Open Hardware Monitor software [34] was used in Intel Core i9, where power consumption was averaged over 100.000 inferences. In NVIDIA Xavier, the GPU was

in “15 W DESKTOP” mode and power consumption was read from GPU power rails via I2C [35]. As for NVIDIA Jetson TX2, the power mode was “MAXP ARM CORE” and the same technique as Xavier was used to get GPU’s power information. Raspberry PI’s input current was measured using a clamp meter and the input voltage was assumed to be constant at 5 V. Idle current was subtracted from the total input current in order to get the dynamic power. Finally, for ZedBoard, the current was measured over the current sense resistor of 10mΩ and the power was calculated assuming a constant voltage of 12V [36]. Both speed and energy efficiency were measured the same way for unoptimised as well as optimised neural networks.

In total, we benchmarked three models on six different hardware platforms, i.e. we did 18 benchmark experiments to evaluate the technology. The models we have used for our benchmarks are **VGG16** [37] and **ResNet18** [38] targeting the image classification dataset Cifar-10 [39] as well as **YoloV3** [40] used for object detection on the popular COCO-dataset [41]. These models are commonly used as the main component in image/video related applications either as a backbone or, in the case of YoloV3, as a complete end-to-end object detector. We consider accuracy drop of less than one percent as being negligible.

6.5 Results

The results from the optimisation done on the different models on several hardware platforms are presented in Figure 31 to Figure 36, comparing both speedup (measured in frames per second) and increased energy efficiency (measured in inferences per Joule). As observed in the results the optimisations done consistently increase the performance of models on a diverse set of hardware. The optimization results vary with the model and hardware platform. We could observe a minimum speed up of 1.5x and 2.3x increase in energy efficiency. Also, several model-hardware combinations show above 10x improvements.

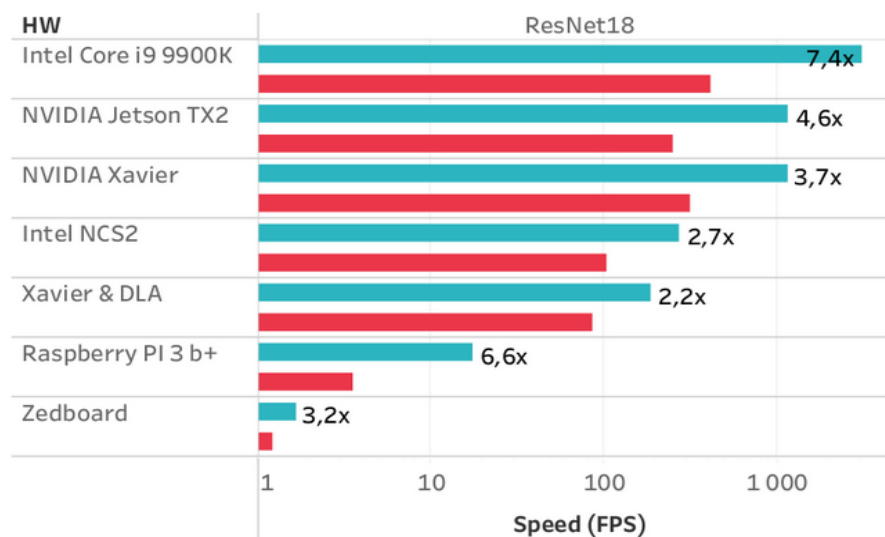


Figure 31: Execution time improvements comparing EmbeDL (blue) with baseline model (red) for VGG16 on six different hardware platforms

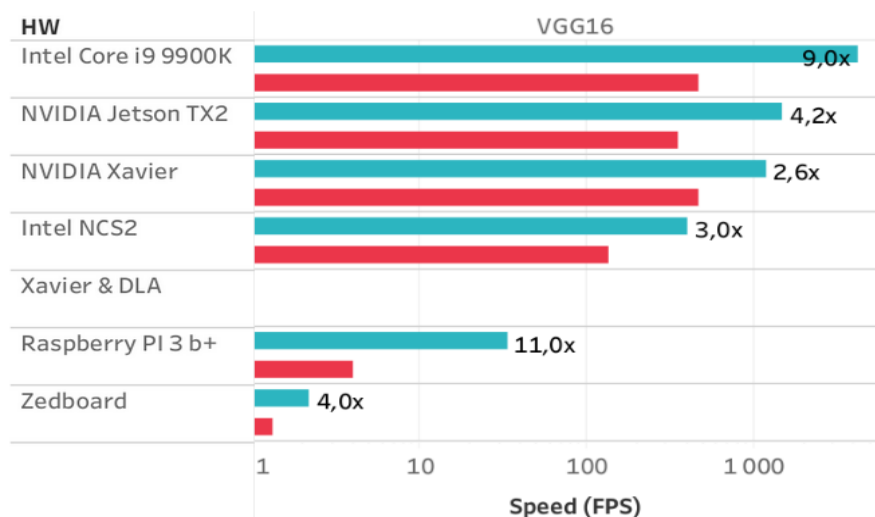


Figure 32: Execution time improvements comparing EmbeDL (blue) with baseline model (red) for Resnet18 on six different hardware platforms

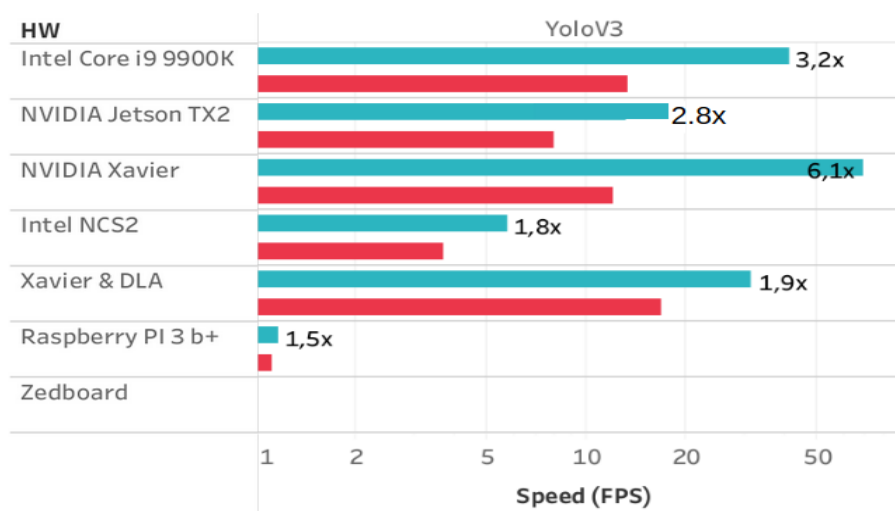


Figure 33: Execution time improvements comparing EmbeDL (blue) with baseline model (red) for YOLOv3 on six different hardware platforms

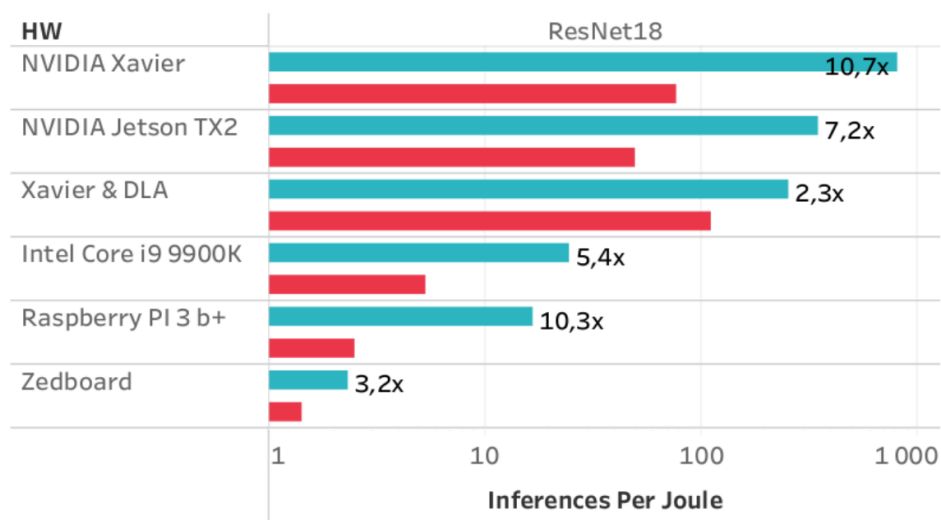


Figure 34: Energy efficiency improvements comparing EmbeDL (blue) with baseline model (red) for Resnet18 on six different hardware platforms

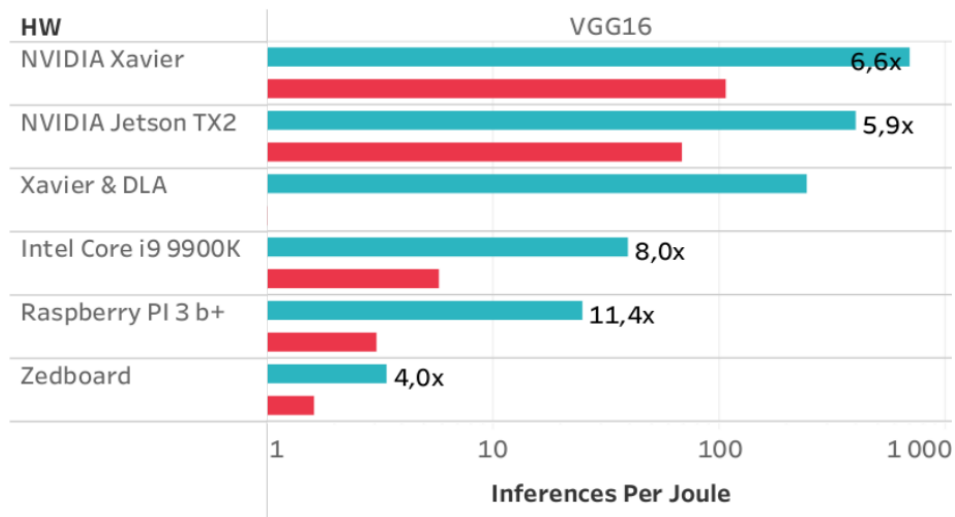


Figure 35: Energy efficiency improvements comparing EmbeDL (blue) with baseline model (red) for VGG16 on six different hardware platforms

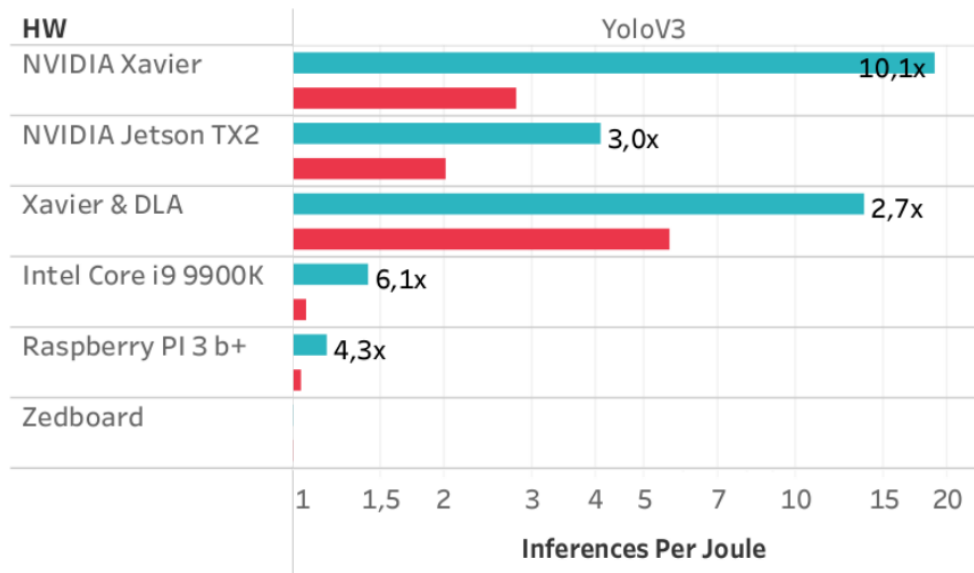


Figure 36: Energy efficiency improvements comparing EmbeDL (blue) with baseline model (red) for YOLOv3 on six different hardware platforms.

For increased readability, the relative improvements of both speed-up and energy efficiency are presented in Table 12 and Table 13. From these tables, the average improvement in speed-up is 4.3x and in energy efficiency is 6.3x.

Speed-Up (X)	ResNet18	VGG16	YoloV3	Average
Intel Core i9 9900K	7.4	9.0	3.2	6.6
NVIDIA Jetson TX2	4.6	4.2	2.8	3.9
NVIDIA Xavier	3.7	2.6	6.1	4.1
Xavier & DLA	2.2		1.9	2.1
Intel NCS2	2.7	3.0	1.8	2.5
Raspberry pi 3 b+	6.6	11.0	1.5	6.4
Zedboard	3.2	4.0		3.6
Average	4.3	5.6	2.9	4.3

Table 12: Summary of the speed-ups measured in the use case experiments, including average improvements for each hardware platform and model

Energy (X)	ResNet18	VGG16	YoloV3	Average
Intel Core i9 9900K	5.4	8.0	6.1	6.5
NVIDIA Jetson TX2	7.2	5.9	3.0	5.4
NVIDIA Xavier	10.7	6.6	10.1	9.1
Xavier & DLA	2.3		2.7	2.5
Raspberry pi 3 b+	10.3	11.4	4.3	8.7
Zedboard	3.2	4.0		3.6
Average	6.5	7.2	5.3	6.3

Table 13: Summary of the energy efficiency improvements measured in the use case experiments, including average improvements for each hardware platform and model

Both speedup and energy efficiency vary between hardware, which is expected since different hardware comes with its own set of memory- and compute constraints. Also, YoloV3 wasn't evaluated on Zedboard due to hardware limitations and the unoptimized version of VGG16 did not run on Xavier & DLA possibly due to lack of memory but we still are not certain. We were also not able to measure the energy consumption of the Intel Neural Compute Stick.



Figure 37: Optimization leads to reduced latency which improves a system's real-time capabilities. This can be seen in the figure comparing optimized vs original YOLOv3 object detection model. Left: Optimized by EmbeDL Deep Learning Optimization Engine. Right: Original model. Run on Nvidia Xavier using Nvidia TensorRT-7.1.3.

Due to sequential execution, latency is improved with the same factor as speedup. In Figure 37 we can also qualitatively assess the importance of low latency for dynamic vision-based applications. The bounding boxes are drawn offset in the video with the corresponding latency. The optimized system is doing a much better job following a moving object.

7 Secure IoT Gateway

The Internet of Things (IoT) is on everyone's lips. There are completed products or instructions for do-it-yourself projects for nearly all interest groups. The growing number a diversity of IoT devices has already many effects on our lives as more and more of our devices are communicating among each other, no matter if we are a private or commercial user. Still, the security of IoT is often unattended for different reasons: it seems to be unnecessary for so small devices or too complex, to name just two often heard excuses.

To provide a solution to reduce the complexity of IoT security, the Secure IoT Gateway was developed. See <https://embedded.christmann.info/secure-iot-gateway/> for a customer-focused product description.

Talking about Industry 4.0 and Internet of Things, the security of communication should be a main focus. Many devices are communicating among each other or to a central server/cloud. This could be in a local area network (LAN) like a company network as well as a wide area network (WAN) like the internet. No matter whether an IoT device is located in a local or wide area network, the communication through the network needs to be secure against any attacks. As any part of the communication can be attacked, an effective security concept is needed, covering the whole path between sender and receiver. The security of IoT and/or remote network devices is really complex, so the Secure IoT Gateway supports the Smart Home use cases by simplifying this complexity. The Secure IoT Gateway is a special use case in the LEGaTO project as its main goal is not to optimize the energy efficiency as the other ones but to reduce the complexity of IoT security and add another security layer to the use case.

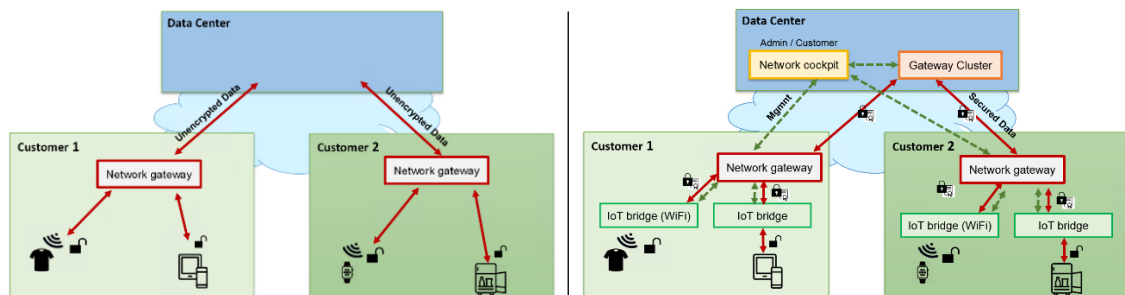


Figure 38: Unsecured (left) versus secured (right) network scenario, using the Secure IoT Gateway components

Figure 38 describes a typical basic network and compares the unsecured and secured setup, using the developed components of the Secure IoT Gateway which provides major security improvements by encrypting and controlling traffic flow from IoT devices. The system focuses on three main segments: Encryption, communication control and interconnectivity. The Secure IoT Gateway is centred on an easy-to-use web application, which allows the complete control over so called "IoT Bridges" and "IoT Gateways" – the hardware devices that ensure encrypted and controlled network traffic. Interconnectivity is achieved in the scope of cross-network communication, allowing IoT devices to communicate with each other even if they are in separate networks. VPN is the underlying technology for communication encryption and interconnectivity. Communication control is achieved with an in-house built version of the open-source Firewall OPNsense [42]. The different components and technologies that make up the Secure IoT Gateway are further described in section 7.1.

7.1 Components

The Secure IoT Gateway is an application composed of four components working together to create a secure communication.

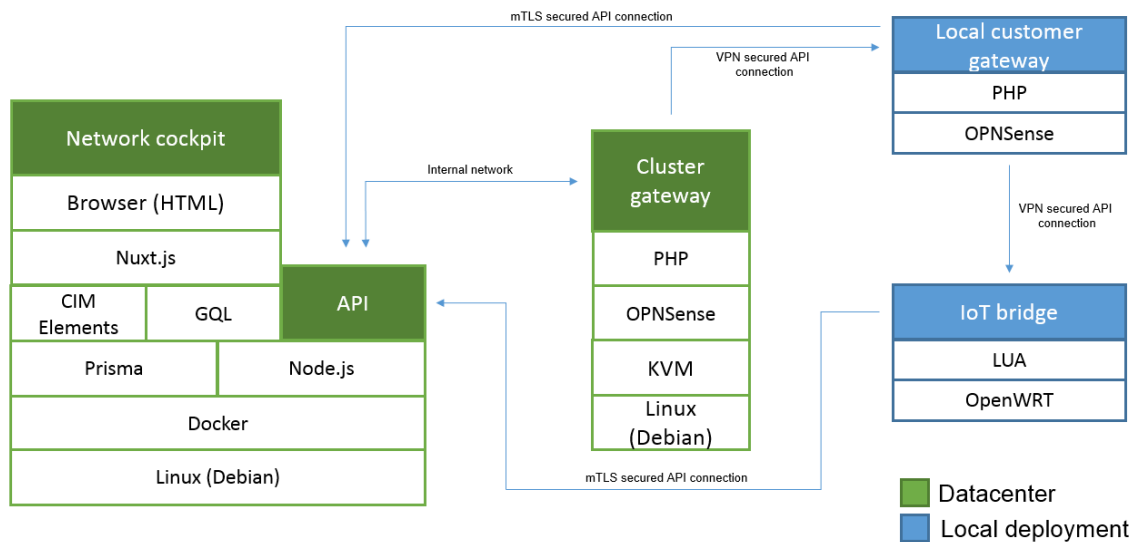


Figure 39: Technologies, languages and frameworks that make up the Secure IoT Gateway

As shown in Figure 39, VPN is the main underlying security technology besides Mutual Transport Layer (mTLS) for secured management communication. Instead of directly accessing the network, IoT devices are now forced to communicate over the newly established VPN connections between IoT Bridges – the VPN entry point for IoT devices – and the Local Gateway, which functions as the VPN endpoint supplier. The Secure IoT Gateway solution is split into two operating environments. One part is the local deployment where devices are placed at the customer's site (IoT Bridges and Local Gateway). In order to manage these locally placed devices and their VPN connections, there is a counterpart in the datacenter, consisting of the Network Cockpit and Cluster Gateway.

The purpose and functionality of the used frameworks as shown in Figure 39 is further described in section 7.1.4 - Underlying Technologies.

Secure Communication through mTLS

A secure and reliable remote configuration of the Secure IoT Gateway components is achieved through HTTPS requests. Those requests are sent through a VPN tunnel, thus protected and encrypted with two methods. In order to authenticate and validate the different devices at the Network Cockpit, mutual TLS connection (two-way authentication) is used. Mutual TLS behaves like a normal TLS authentication used in HTTPS, but instead of only validating the server side, mutual TLS also authenticates the client. When properly handshaked, the following traffic is encrypted and both identities are confirmed. To ensure a functional mutual TLS connection, client certificates are necessary which are created for every device during the provisioning process, further described in Figure 41.

7.1.1 IoT Bridge

The **IoT Bridge** is a small local device running an embedded Linux system specialised for network activities, OpenWRT [43]. One IoT Bridge is placed directly before any IoT device that should be secured. The connection will be encrypted on-the-fly and can be established with a Gateway Cluster or a Network Gateway. As a component in a local area network, the IoT Bridge is not multi-customer capable.

The main component of the Secure IoT Gateway is the IoT Bridge. It works as the network entry point for IoT devices and ensures a secured connection by encrypting traffic with OpenVPN. OpenWRT is an open-source, highly customizable, Linux based router firmware that runs on small embedded devices like the listed product line in Figure 40. These ARM-based embedded hardware components were chosen because they offer sufficient network throughput and encryption performance in a small form factor. To secure an IoT device, the bridge must be connected in-between the local network and the IoT device itself. This ensures that the whole traffic that is sent and received by the IoT device is controlled and encrypted. In order to allow remote configuration of the IoT Bridge, we have extended the functionality of OpenWRT with several extensions written in the lightweight scripting language Lua [44]. Those extensions enable secure remote access to OpenWRT's Unified Configuration Interface (UCI) and are included in the custom image of the IoT Bridge that is compiled, pre-configured and bundled from source within our continuous integration environment.




Device	IoT Bridge 10	IoT Bridge 50 WiFi	IoT Bridge 100
Picture			
SoC	Atheros9331 400MHz	Allwinner H3 4 x 1,2 Ghz	Rockchip RK3328 4 x 1,3 Ghz
Power	< 1.5 W	< 10 W	< 10 W
VPN Speed	9 Mbit/s	55 Mbit/s	95 Mbit/s
WiFi	✓	✓	X
Flash	Integrated	Integrated	SD-card

Figure 40: IoT Bridge hardware options

The hardware options are tuned for different use-case scenarios. The most noticeable differences regarding the three hardware options is the encryption speed and the resulting network throughput. While the IoT Bridge 10 is mostly used for low-bandwidth applications like IoT sensors, IoT Bridge 50 and 100 are more suitable for mid- and high-throughput IoT applications like video streaming or remote access use-cases.

Provisioning

All those programs and pre-configurations are not delivered with the hardware from the original manufacturer, but have to be put on the devices by us. To accelerate the provisioning process, a program was developed to automatically make all the steps needed for a delivery of a device to a customer which is described in Figure 41.

First, the device is flashed with a custom version of OpenWRT, containing our extensions, necessary Unix packages, custom time-series jobs, services and a base configuration. Now the custom provisioning software comes into play.

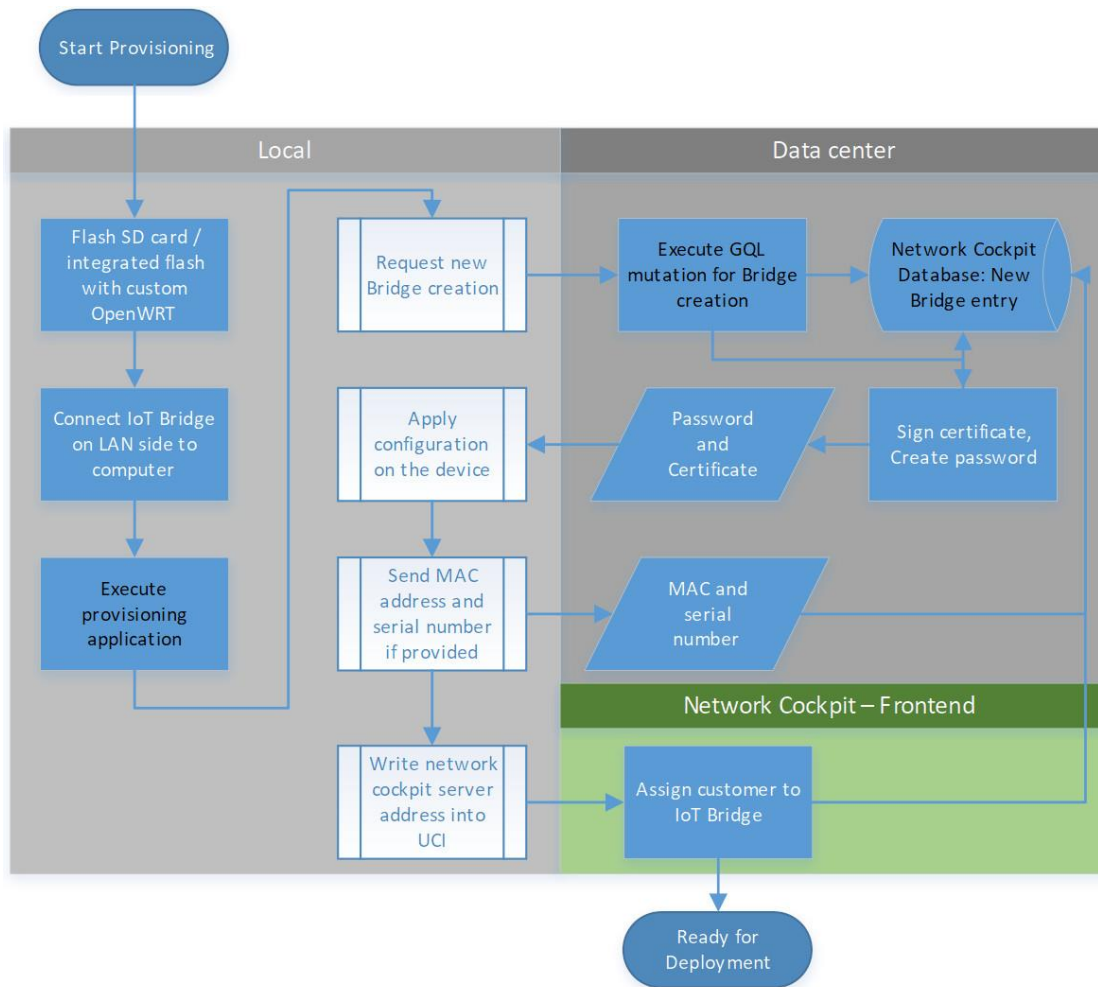


Figure 41: Flowchart of the provisioning process

The provisioning program requests an individual password and client certificate from the Network Cockpit API. The API is now creating a new SSL certificate, signed by a certificate authority located on the Network Cockpit server. At the same time, a new database record is created which contains the previously mentioned client certificate and password for the new IoT Bridge. The newly created password and certificate are now sent back to the provisioning program which passes it onto the IoT Bridge. MAC Address and Serial number (if provided) will be read from the system and sent to the Network Cockpit for later identification. Before completion, the API server address and configuration parameters for the OpenWrt extensions are set. The IoT Bridge is ready to be shipped to a customer.

7.1.2 Local Gateway

The **Local Gateway** is a local network component and therefore not multi-customer capable. The Network Gateway is a server with several Ethernet ports and runs an OPNsense instance. It allows OpenVPN secured connections to the IoT Bridges. All the communications between the components can be regulated by rules.

When it comes to deployment of IoT Bridges at a customer's site, a Local Gateway is necessary to provide VPN endpoints for the IoT Bridges. Besides Local Gateways, the Cluster Gateway also plays an essential role in connecting multiple devices from different company locations. The Local Gateway itself has a VPN connection to the Cluster Gateway, allowing traffic routing to multiple customer sites. Both gateway types run on OPNsense – an open-source firewall operating system based on FreeBSD and in-built OpenVPN capabilities. To seamlessly integrate the Local Gateway and Cluster Gateway into the management, several OPNsense extensions were developed to enable remote configuration of firewall rules and OpenVPN servers / clients. Besides configuring rules and servers, these extensions are capable of sending system information and analytic data.

7.1.3 Cluster Gateway

The **Cluster Gateway** is a WAN component and will be hosted by a systems house and runs on an OPNsense instance. By doing this, the Cluster Gateway is the central component where other Local Gateways connect to. Therefore, the Cluster Gateway provides a secure network connection to the datacentre/server for the customer's Local Gateways. To handle several customers, the Gateway Cluster is multi-customer capable.

7.1.4 Network Cockpit

The **Network Cockpit** is the central management interface that users (normally the administrator) will use to configure and monitor the IoT Bridges and Local Gateways. Normally, the user just sees and handles this part of the Secure IoT Gateway, which hides the configuration complexity of the underlying security processes and all different software and hardware parts. The Network Cockpit will be provided as a Cloud Service.

As a web application, the Network Cockpit comes with access authorization. So just authorized users are allowed to use the Network Cockpit. The authorized users will be furthermore differentiated in several permission roles:

- 1) **Users:** A User has access to the information and forms displayed in the network cockpit. The User only gains management permissions for specific devices, which can be assigned by the admin accounts. An account with this permission spectrum can be created and managed by a customer admin or the super admin and is intended for monitoring customer specific IoT bridges, IoT devices and VPN connections. The user could be any IT personnel or e.g. a head of department.
- 2) **Customer Admin:** Customer Admins have full configuration control over the company's devices such as IoT Bridges and Local Gateways. In addition, to configure the existing devices, the admin is allowed to create new accounts, new devices and Secure IoT Gateway components like IoT Bridges. Every new customer gets their own customer admin account – created by the super admin. Normally, this would be an IT administrator of the customers company.
- 3) **Admin:** The Admin has all permissions like the Customer Admin: creation and management of users, devices and Secure IoT Gateway components. The Admin maintains the Gateway Network and is able to see all customers with their secured connection environments which are hosted by the systems house. This account is designed for technicians working in the systems house.

- 4) **Super Admin:** The Super Admin has complete control over the devices and customer accounts. Compared to the normal Admin, the Super Admin has full access to the user management system and is capable of creating and maintaining customer admin and admin accounts. Besides admin account creation, the Super Admin is also the only account allowed assigning newly provisioned IoT Bridges to customers. This account is also designed for technicians working in the systems house but more focused on device provisioning and their customer assignment rather than technical support.

The Network Cockpit is built around some core features which are also reflected inside the main navigation menu. The “Dashboard” provides a good overview of all important activities and devices as well as some statistics. Monitoring and configuration of involved IoT Bridges and Local Gateways can be found under the “My Devices” menu item on the left side. Setting up communication rules for involved IoT devices is located under the menu item “Rules”. Additionally, you can find a “User Management” and a “Factory” page for internal provisioning use inside the main menu.

After logging into the Network Cockpit, the user sees the Dashboard (see Figure 42) which consists of multiple widgets that can dynamically be arranged in different widget slots.

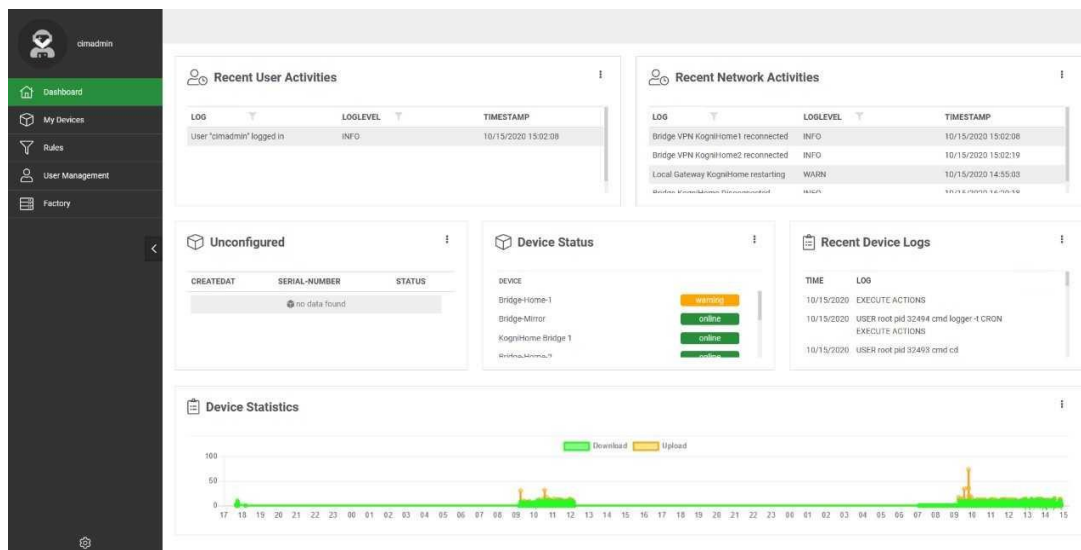


Figure 42: Network Cockpit dashboard with a broad overview of the system

These slots are prefilled with dashboard widgets, but the user has the option to configure the dashboard individually. The widget shop allows the user to select the widget of choice and insert it into the dashboard slot. Currently, the widget shop offers the following options:

- Recent user activities
- Recent network activities
- Un-configured devices
- Device status
- Device logs
- Device statistics

Every Secure IoT Gateway component regularly sends monitoring data about the system it runs on. The Monitoring page presents this data as real-time information with CPU, RAM, bandwidth usage, system logs, connection status and some general information about the selected device. The information as seen in Figure 43 are specific for one selected device which can be found under the “My devices” menu, and then clicking on “Monitoring”.

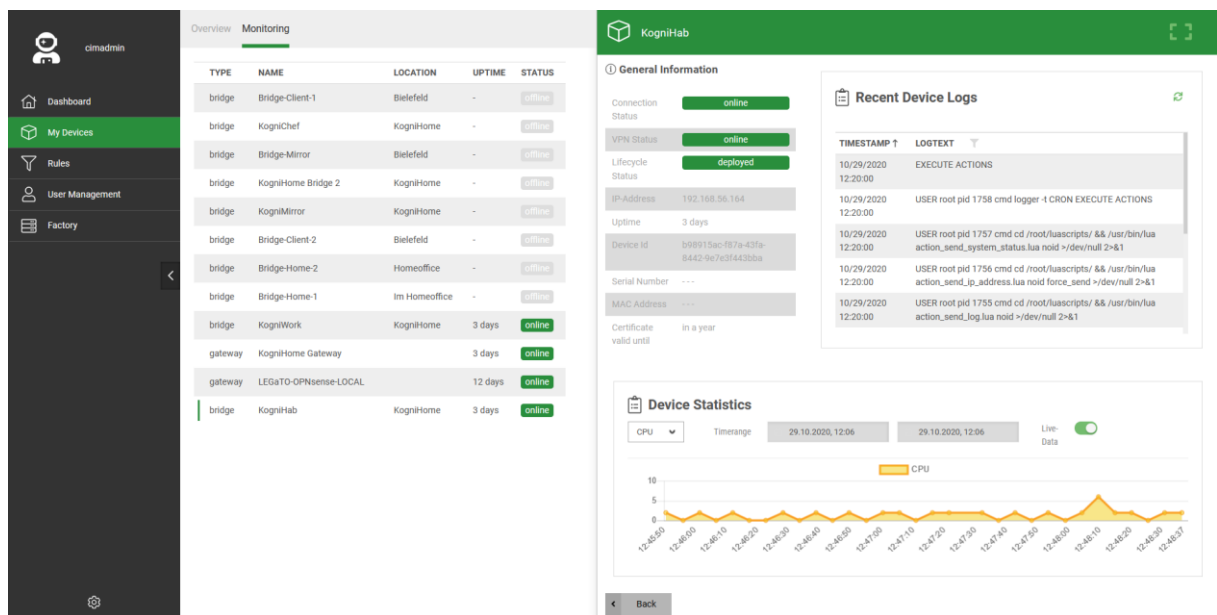


Figure 43: Network Cockpit monitoring tab that shows device related statistics

“My Devices” also provides the “Overview” page, which is designed for a straight forward configuration of network components as seen in Figure 44.

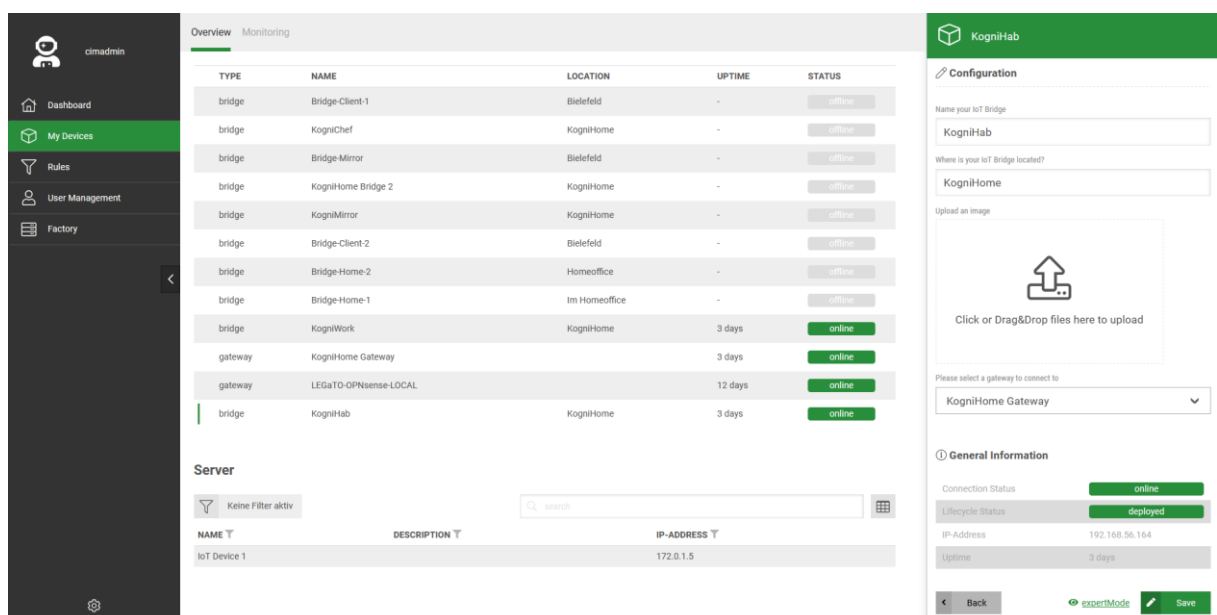


Figure 44: Network Cockpit device overview for configurations

Upon double-clicking a network component, the sidebar opens and reveals the configuration options for the selected device. The first step of configuring new devices is always this page, because it allows the user to name IoT Bridges and connect it to a Local Gateway. Besides naming and connection management, this view also offers basic monitoring capabilities (uptime and status), a possibility to define the location of the IoT Bridge, adding a photo and general information like the current IP address.

When deployed and connected, an IoT Bridge has no communication rights per default. This ensures maximum security because all ports are blocked and no possible attack vectors are available inside the network. To allow certain services and connections, the user has to change the ruleset of specific devices in the “Rules” page as seen in Figure 45.

Figure 45: Network Cockpit rule configuration for an IoT Bridge

The Rule Configuration differentiates between IoT Bridges and protocols. By selecting „Another IoT Bridge“ as traffic destination, the IoT device behind the specified IoT Bridge is able to communicate with the target IoT Device. It is also possible to allow specific protocols and services like DHCP, DNS and HTTP / HTTPS to pass through by selecting the desired entry inside the „Traffic destination“ menu.

The “Factory” page handles the customer assignment of freshly provisioned IoT Bridges. This page is only accessible to the systems house super-admin.

In order to create new users, admins have access to the “User Management” page, which allows customers to create sub-user accounts. The super admin uses this page mainly for creating new customer-admin accounts.

Underlying Technologies

The Network Cockpit runs on a Linux System and multiple Docker Containers provide the basis and making the application available as a web application. The Network Cockpit is realized using HTML, CSS 3 und ECMAScript 6. Within the different languages, the following frameworks are used:

- Nuxt.js (extended Vue.js frontend Framework) [45]
- Bulma (Frontend Library used by Nuxt.js) [46]
- CIM Elements (In-house built backend/frontend node packages from christmann)
- Prisma (Database management based on PostgreSQL) [47]
- Express.js (Node.js webserver) [48]
- GraphQL Nexus (JavaScript GraphQL Schema Builder) [49]

Nuxt.js is an open source web application framework, based on Vue.js. Nuxt.js makes the configuration and setup of web applications very simple. As it can build web applications using

server-rendering, it is more optimized for search engines. At the same time, the created web application acts like a Single Page Application with its interactivity so the user experience isn't suffering caused by long loading times.

Bulma is an open source CSS library that provides customizable and easy-to-use frontend elements that can be integrated via the nuxt.js framework. The library consists of multiple form elements, buttons, tables, navigation bars and other visual interface components.

CIM Elements is a collection of in-house developed node.js packages, developed by christmann, that provide a variety of different features such as user management and authentication, permission management and also UI elements like tables and page layouts.

Prisma is an open-source database toolkit. It works as an object-relational mapping and makes database access to PostgreSQL easy with an auto-generated query builder for Typescript & Node.js.

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

GraphQL Nexus is a framework which automatically generates GraphQL resolvers for CRUD-operations for database datasets. GraphQL is an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data.

7.2 Integration in the Smart Home use-case

The Secure IoT Gateway was integrated and tested in real-life usage in two different existing environments. First, we installed the system into the Bielefeld University network of the CITEC building with the goal of securing the publicly accessible Smart Mirror project, see section 3. This led to further development in the scope of a renewed hardware revision for the second integration in the KogniHome, a research flat for smart devices and assisted living in the Bielefeld city centre, see section 7.2.2.

7.2.1 Smart Mirror prototype

The initial test of the system in a real-world application was done in partnership with the Bielefeld University inside the CITEC building. The system was installed with the main focus of securing connections between the publicly accessible Smart Mirror prototype and the remote access client computers of the researchers. Some challenges had to overcome before we could start with the installation. The CITEC network required authentication via the IEEE 802.1X protocol, which was not supported by the IoT Bridges per default. The test revealed some missing possibilities in the remote configuration management of the IoT Bridges and one key feature – the Rule Configuration – was still in development and not ready for usage yet. Nevertheless, the system worked after some manual configuration. Later tests and benchmarks showed that the chosen hardware devices for the IoT Bridges were too slow to handle the remote desktop connections, so a new hardware selection became necessary.

7.2.2 KogniHome research flat

In the second integration, we installed the Secure IoT Gateway in the KogniHome research flat in the Bielefeld city centre. The KogniHome project aims to develop and showcase new smart home

technologies that allow technology-assisted living for all age generations and people with disabilities [50]. Multiple companies are part of the KogniHome project and contribute with their latest home assistant technology. In order to test and showcase the different technologies, a flat in Bielefeld was equipped with the latest smart home technologies, allowing a glimpse into the future of technology-assisted living standards. The four IoT devices KogniMirror, KogniChef, KogniWork and KogniHab were equipped with an IoT Bridge 50, therefore secured and controlled in their network communication. The IoT Bridge 50 is part of the new hardware revision and allows for higher VPN encryption speeds and better overall stability and performance. All traffic between the devices is now encrypted. Furthermore, access to unused network ports was denied by the rule configuration provided by the Network Cockpit.

Besides IoT Bridges, we also installed a Local Gateway to provide VPN endpoints for the devices. To allow communication via MQTT (Protocol for IoT messaging [51]) and RSB (Robotics Service Bus – for scalable integration of robotic systems [52]), some manual configuration on the Local Gateway was necessary, because the Network Cockpit rule configuration currently does not support the selection of these protocols. Services like ICMP, DHCP, DNS and HTTP/HTTPS, on the other hand, can dynamically be configured by the Rule Configuration.

In preparation of the integration, the new network topology and the integration points of the four IoT Bridges and the Local Gateway were discussed as seen in Figure 46. The developers and operators of the KogniHome were not able to change the IP addresses of the IoT devices that should be secured. Therefore, the development of a new feature was necessary to get working connections inside the existing KogniHome network structure without creating new subnets which would require new IP addresses. Rather than using a conventional OpenVPN routing setup, we decided to use a bridging configuration, making integration into the existing network structure possible. In contrast to the CITEC integration, this VPN network mode operates on layer 2 and does not create new subnets for the devices connected behind an IoT Bridge. The new feature was implemented into the Network Cockpit and allows customers to set their preferred network mode now.

To allow a secure way of remote access, an IoT Bridge was configured to directly communicate with the Cluster Gateway, thus allowing encrypted network access over the internet. This is labelled “Private Home” in Figure 46.

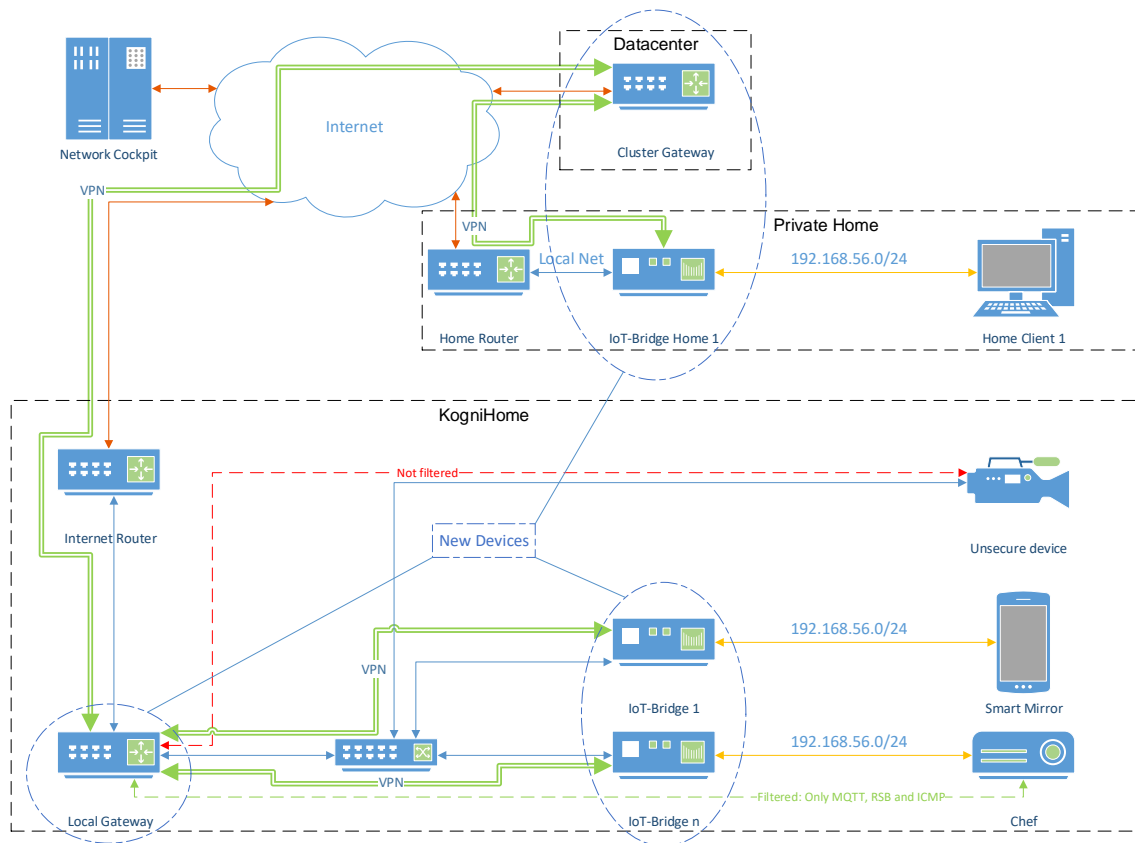


Figure 46: KogniHome network with IoT Secure Gateway integration

7.3 Benchmarks

In D5.2 [2] the benchmark between OpenVPN and Wireguard was explained. Because of the better scalability with parallel tunnels and the higher resulting overall VPN throughput the decision was made in favor of OpenVPN. Also the CPU utilization is higher with Wireguard. This could result in problems on the Local Gateway when a lot of VPN connections have to be served.

PCIe crypto card vs CPU

In addition to that benchmark, it was tested if a crypto accelerator could speed up the VPN connection. While Wireguard is not capable to use such an accelerator, OpenVPN can use it. The test should originally be done with the system used for the VPN connections which is OPNsense. Unfortunately, the Intel QAT driver that is used by the crypto card PE3ISLBEL from Silicom didn't work with the FreeBSD used by OPNsense. And the current default hardware used by Christmann for the Local Gateway currently doesn't support and boot with the crypto card. As an alternative, a test was done under CentOS 8 with OpenSSL to find out if the card would bring any advantage. The test was done with an AMD EPYC 7262 CPU running at 3,2 GHz which can be well used in large installations. The comparison system was the PE3ISLBEL crypto card and with an Intel Atom C2558 CPU running at 2,40 GHz as it is used in the default Local Gateway. The benchmark was done with OpenSSL version 1.1.1c that was built to use the Intel QAT engine. The command "openssl speed" was used to utilize only the CPU and "openssl speed -engine qatengine" to use the crypto card during the benchmark. The following table shows the comparison of throughput with different block sizes for the aes-128-cbc-hmac-sha1 algorithm.

The results can be seen in Figure 47. They show that the crypto card brings a big advantage when a power efficient CPU like the Intel Atom is used. Still, a fast CPU like the tested AMD EPYC can outperform the crypto when using more than two processes.

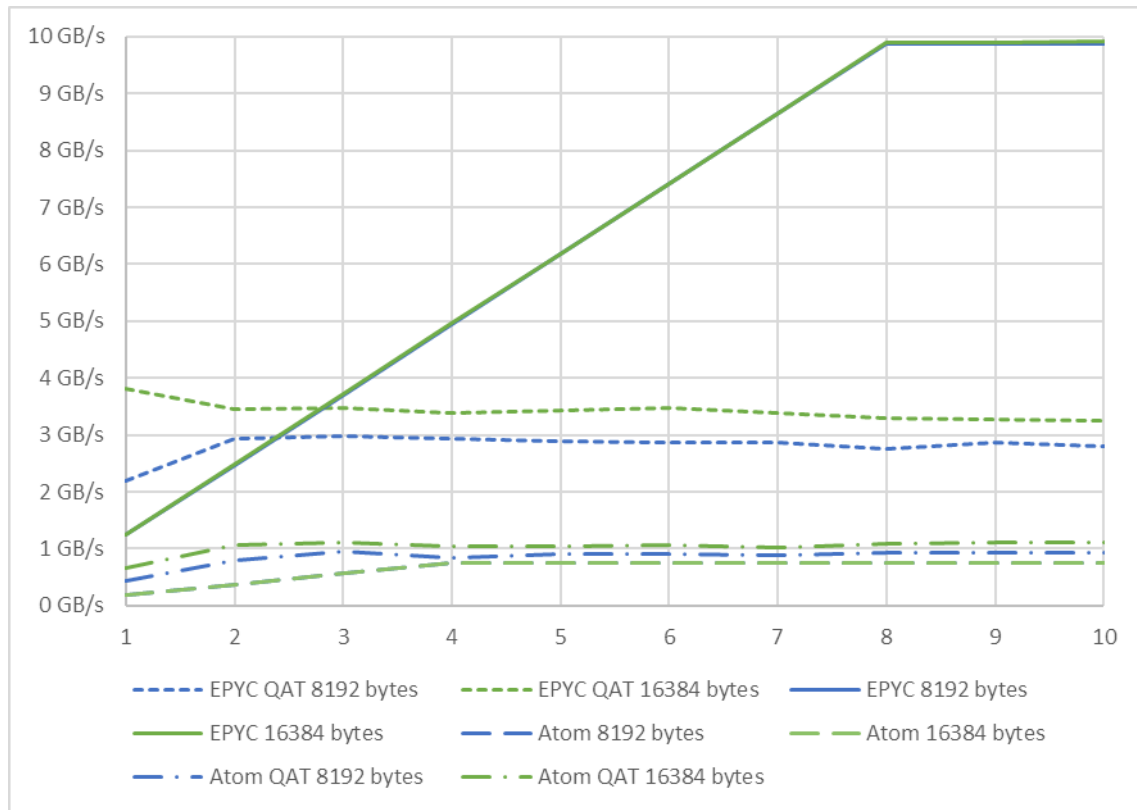


Figure 47: Comparison of aes-128-cbc-hmac-sha1 between AMD EPYC 7262, Intel Atom C2558 @2.40GHz and PE3/SLBEL crypto card with QAT driver

Table 14 shows that the crypto card only accelerates key signing operations for private keys and some verification operations of public keys with the recent firmware. Some operations are not supported by the crypto card. This is the case where there is no difference between the crypto card and the CPU. But the QAT engine redirects those operations back to the CPU.

algorithm \ process	sign duration	signed per second	verify duration	verified per second
rsa 512 bits	154,2%	157,1%	15,4%	18,2%
rsa 1024 bits	119,5%	118,9%	46,2%	46,6%
rsa 2048 bits	419,0%	418,1%	48,6%	48,4%
rsa 3072 bits	494,4%	494,6%	119,4%	118,9%
rsa 4096 bits	628,8%	629,3%	123,1%	123,0%
rsa 7680 bits	99,8%	99,6%	100,0%	100,0%
rsa 15360 bits	100,0%	100,0%	100,0%	100,0%
dsa 512 bits	75,3%	75,8%	100,0%	100,4%
dsa 1024 bits	132,0%	131,8%	172,1%	171,3%
dsa 2048 bits	439,0%	439,4%	98,7%	98,4%
224 bits ecdsa (nistp224)	50,0%	34,5%	33,3%	53,6%
256 bits ecdsa (nistp256)	0,0%	14,3%	33,3%	26,4%

384 bits ecdsa (nistp384)	500,0%	409,9%	140,0%	149,9%
521 bits ecdsa (nistp521)	100,0%	85,5%	83,3%	92,4%
253 bits EdDSA (Ed25519)	100,0%	99,8%	100,0%	101,3%
456 bits EdDSA (Ed448)	100,0%	99,8%	100,0%	100,6%
operation algorithm	operations per second			
224 bits ecdh (nistp224)	70,5%			
256 bits ecdh (nistp256)	35,0%			
384 bits ecdh (nistp384)	456,5%			
521 bits ecdh (nistp521)	107,0%			
253 bits ecdh (X25519)	79,8%			
448 bits ecdh (X448)	568,6%			

Table 14: Comparison of OpenSSL test with AMD EPYC 7262 vs. PE3ISLBEL crypto accelerator card

Concluding, it can be said that using the crypto card makes sense when many IoT Bridges are used on one Local Gateway. This will be further evaluated in future projects, also the integration of the QAT driver with OPNsense must be worked on.

Comparison of different IoT Bridge devices

Three different hardware device to be used as IoT Bridge were compared and benchmarked to find out the maximum encrypted VPN throughput. The tests were done using AES-128-CBC encrypted VPNs running iperf3. In the test, the different IoT Bridges were connected to a fast Local Gateway. Then, two computers were connected, one to the Local Gateway and one to the IoT Bridge to measure the throughput and the ends, as displayed in Figure 48.

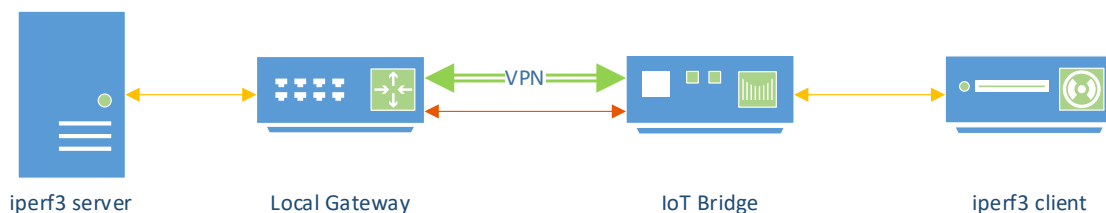


Figure 48: IoT Bridge benchmarking setup

IoT Bridge 100	iperf3 -c 192.168.2.200 -O 5 -P 3 -t 60		iperf3 -c 192.168.2.200 -O 5 -P 3 -t 60 -u -b 1G	
	TCP		UDP	
60 sec ▼	Transfer	Bandwith	Transfer	Bandwith
Sending	515 MBytes	72,0 Mb/s	777 MBytes	109,0 Mb/s
Receiving	612 MBytes	85,6 Mb/s	708 MBytes	98,9 Mb/s

IoT Bridge 50	iperf3 -c 192.168.2.200 -O 5 -P 3 -t 60		iperf3 -c 192.168.2.200 -O 5 -P 3 -t 60 -u -b 1G	
	TCP		UDP	
60 sec ▼	Transfer	Bandwith	Transfer	Bandwith
Sending	307 MBytes	42,9 Mb/s	298 MBytes	41,7 Mb/s
Receiving	361 MBytes	50,4 Mb/s	438 MBytes	61,2 Mb/s

IoT Bridge 10	iperf3 -c 192.168.2.200 -O 5 -P 3 -t 60		iperf3 -c 192.168.2.200 -O 5 -P 3 -t 60 -u -b 4M	
	TCP		UDP	
60 sec ▼	Transfer	Bandwith	Transfer	Bandwith
Sending	84.9 MBytes	11,9 Mbits/sec	64.2 MBytes	9,0 Mbits/sec
Recieving	67.0 MBytes	9,4 Mbits/sec	61.1 MBytes	8,5 Mbits/sec

Table 15: Troughput of IoT Bridges tested with iperf3

After benchmarking the IoT Bridges, the available hardware was devided in three products whose names indicate their VPN performance.

Name	VPN performance	Typical power consumption
IoT Bridge 10	9 Mbit/s	1,2 W
IoT Bridge 50	55 Mbit/s	1,7 W
IoT Bridge 100	95 Mbit/s	2,1 W

Table 16: IoT Bridges VPN performance

Ping, round trip time

As some services like Voice-over-IP are sensitive to the ping, the Round-Trip Time was measured in the local network and to an external server. With an established VPN, the ping time increases only by about 1,9 ms. This allows to use time critical applications like remote services through the VPN of the IoT Bridges without the user noticing a significant delay. An Intel NUC with an Intel Celeron J4005 CPU running at 2,0 GHz and 4GB RAM was used behind an IoT Bridge 10 to test a Windows Remote Desktop (RDP) session. Several programs including the MS Office suite, an internet browser, and the Windows Explorer were tested. There was no noticeable difference between an unencrypted and VPN encrypted connection. The only noticeable difference could be observed when copying a large file into the RDP session. This took more time because of the limited throughput. A speed test in an RDP session running on the Intel NUC was done with the IoT Bridge 10 as seen in Figure 49.

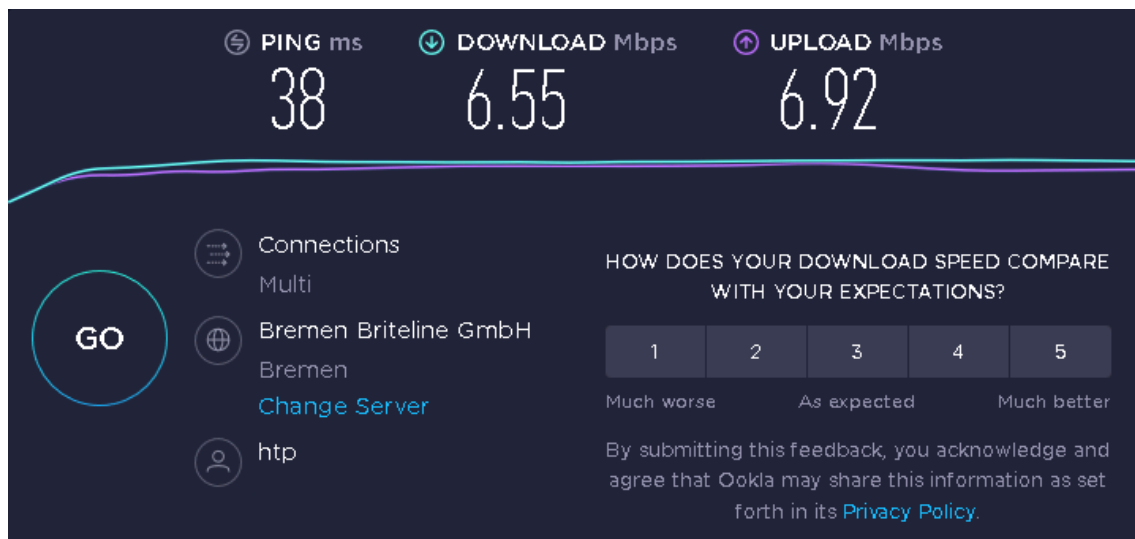


Figure 49: Speed test within a browser over an RDP session through the IoT Bridge 10

Editing Office documents as well as keyboard and mouse input, was very smooth. Video playback over RDP and VPN was not as smooth as without VPN protection. There were some visible artefacts in the video when playing a full HD YouTube video [53] over an RDP session using the IoT Bridge 10. With the IoT Bridge 50 and 100, these artefacts were not visible, and the video played very smoothly.

The results can be seen in Figure 50. The screenshots were taken on the local computer where the RDP session was displayed with exception of the original picture which was taken directly on the RDP host.

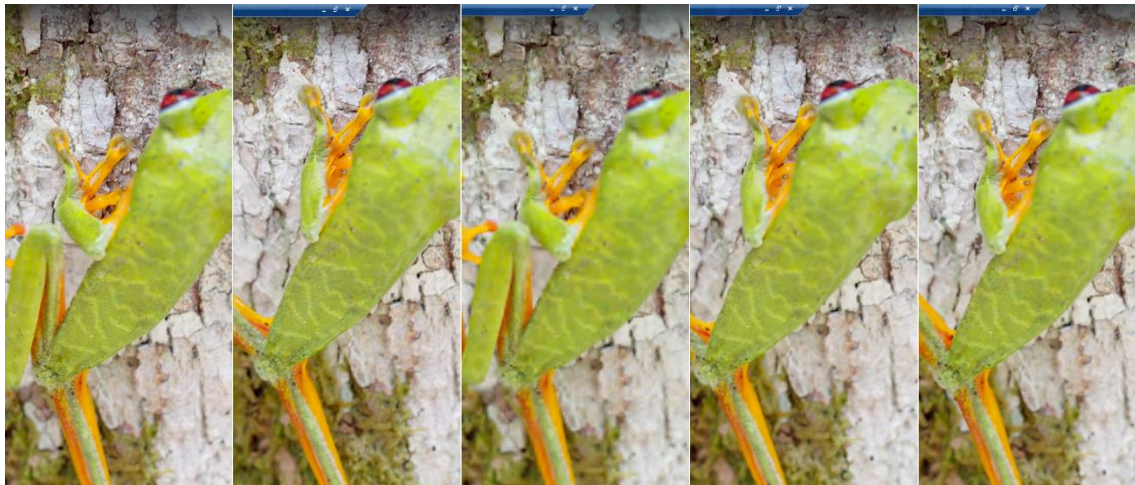


Figure 50: Screenshot of a YouTube video run in 1080p; Left to right: Original, RDP without VPN, RDP with IoT Bridge 10, RDP with IoT Bridge 50, RDP with IoT Bridge 100

With the result of these benchmarks we have learned, that it can be useful to use an extra encryption card together with an energy-efficient CPU as the crypto card only uses 4,9W of power which is less than a powerful CPU would use. This would enable the usage of establishing many VPN tunnels on a cheap and energy-efficient platform. The performance of the cheap IoT Bridges 10 is sufficient to handle IoT services that don't need much bandwidth with a low latency. Such a setup proved to serve as a VPN appliance, e.g. for office environments. If high throughput is expected, a suitable IoT Bridge with more maximum throughput can choose to satisfy those requirements.

8 Overall project integration

In this project, we have integrated several software components in order to achieve a common programming environment around the hardware platforms. See <https://legato-project.eu/software/integration> for the links to the software repositories. We have worked to include all integrations around the OmpSs programming model. This section shows the basic structure of each integration, and it uses the matrix multiplication [54] example to show how it is coded with the integrated version of the programming model. Overall, these are the integrations with OmpSs:

- OmpSs and XiTAO targeting SMP tasking
- OmpSs with support for CUDA and OpenCL kernels
- OmpSs with support for Xilinx FPGAs with High-Level Synthesis and DFiant kernels
- OmpSs with support for Maxeler DFEs (runtime-level only)
- OmpSs with support for Secure SGX tasks (runtime-level only)

Additionally, we have worked on the integration of other components around the OmpSs programming model and on the execution environment:

- Linter tool for OmpSs
- IDE plugin for OmpSs
- RECS_Master and Slurm

The following subsections explain these integrations with further details.

8.1 OmpSs and XiTAO targeting SMP tasking

For the OmpSs and XiTAO integration, we have adapted the two different execution environments so that the application can use both, and each environment has its own resources for task scheduling. Figure 50 shows the toolchain for this environment. The programmer decides which portions of the application should be written with OmpSs tasks or with XiTAO tasks. Once the code is split between the two models, upon starting the application, each runtime system owns its own resources, and schedules tasks on them.

As an easy to understand application sample, we use a matrix multiplication, in such a way that half of the computation is done using OmpSs tasking, and the other half is implemented with XiTAO tasking.

Figure 51 represents the integration of OmpSs with XiTAO in a graphical way, while the code listing in Figure 52 shows the matrix multiplication code example written using OmpSs and XiTAO.

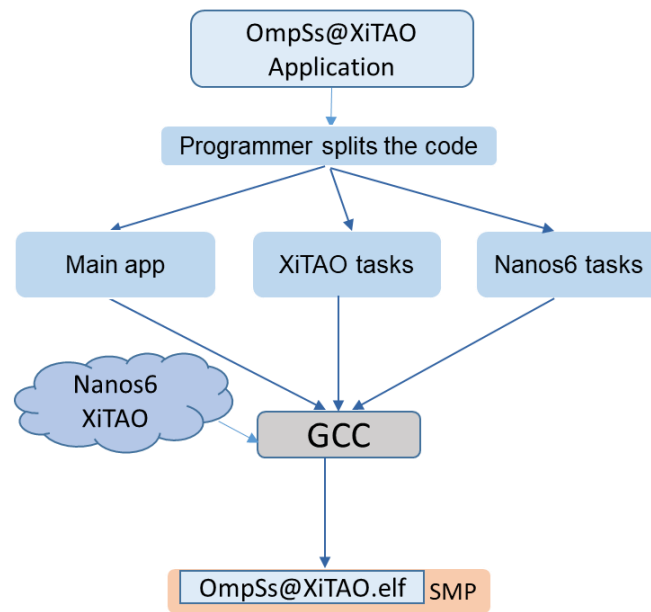


Figure 51: Diagram representation of the integration of OmpSs with XiTAO, targeting SMP tasks

```

// Create the XiTAO graph
MatVecTAO* vm = new MatVecTAO(A_xitao, B, C_xitao, xitao_work_size, N, width);
MatVecCompute(vm, A_omp, B, C_omp, openmp_work_size, N);

...

void MatVecCompute(void * vm, ..., int N)
{
#pragma omp taskwait // OmpSs task to create the parallelism around the OmpSs matmul
{
#pragma omp taskwait for shared(A_omp, C_omp, B, openmp_work_size, N)
for(size_t i = 0; i < openmp_work_size; ++i) {
for(size_t k = 0; k < N; ++k) {
for(size_t j = 0; j < N; ++j) {
C_omp[i * N + j] += A_omp[i * N + k] * B[k * N + j];
}
}
}
}
#pragma omp taskwait // end of the OmpSs work
}

...

// push the TAO Starting point of the XiTAO tasking
GOTAO_push(vm);
//Start the TAODAG execution
GOTAO_start();

#pragma omp taskwait
}

// TAO definition with the XiTAO matmul implementation
class MatVecTAO : public AssemblyTask
{
public:
...

// Inherited pure virtual function that is called by the runtime
// upon executing the TAO.
/*!
\param threadid logical thread id that executes the TAO
This assembly can work totally asynchronously
*/
void execute(int threadid)
{
// int tid = threadid - leader;
size_t li = i++;
while(li < nrows){
for (size_t j = 0; j < N; ++j) {
for(size_t k = 0; k < N; ++k) {
C[li*N + j] += A[li*N + k] * B[k*N + j];
}
}
li = i++;
}
}
}
}

```

Figure 52: Sample matrix multiplication tasks when using OmpSs and XiTAO

8.2 OmpSs with support for CUDA and OpenCL kernels

OmpSs with support for CUDA and OpenCL kernels is able to leverage existing CUDA and OpenCL code without the need to write the CUDA and OpenCL boilerplate. It allows to run the applications on environments with Nvidia GPUs and OpenCL FPGAs. Figure 53 shows the toolchain for this environment. The Mercurium compiler is responsible of generating the code needed to provide the runtime with the necessary data copies, and the wrapper code needed to invoke the CUDA kernels. This wrapper code is later compiled with the Nvidia GPU NVCC compiler [55].

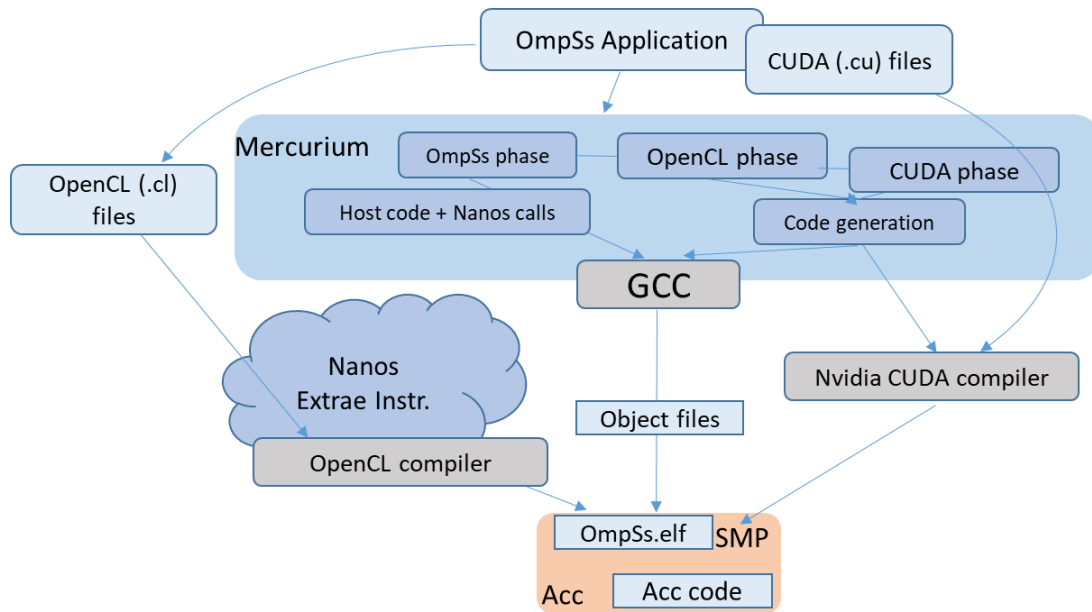


Figure 53: Diagram representation for the integration of OmpSs with CUDA and OpenCL kernels

```
// Matrix multiplication with OmpSs support for CUDA and OpenCL kernels

#pragma omp target device(smp) copy_deps
#pragma omp task in([bsize*bsize]A, [bsize*bsize]B) inout([bsize*bsize]C)
void matrixMult(REAL *C, REAL *A, REAL *B, int wa, int bsize)
{
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, bsize, bsize, bsize,
                1.0f, A, bsize, B, bsize, 1.0f, C, bsize);
}

// CUDA kernel task description
#pragma omp target device(cuda) ndrange(2,NB,NB,BL_SIZE,BL_SIZE) copy_deps
implements(matrixMult)
#pragma omp task inout([NB*NB]C) in([NB*NB]A,[NB*NB]B)
__global__ void matrixMult_cuda(REAL* C, REAL* A, REAL *B, int wA, int wB);

// OpenCL kernel task description
#pragma omp target device(opencl) ndrange(2,NB,NB,BL_SIZE,BL_SIZE) copy_deps
implements(matrixMult)
#pragma omp task inout([NB*NB]C) in([NB*NB]A,[NB*NB]B)
__kernel void matrixMult_opencl(__global REAL* C,__global REAL* A, __global REAL* B,int wA, int
wB);

void matmul(...)
{
    ... // Invokes block matmul and the runtime system decides which device to use for execution
    matrixMult((tileC[i*nDIM+j], tileA[i*1DIM+k], tileB[k*nDIM+j],NB,NB);
}

```

Figure 54: Matrix multiplication with OmpSs support for CUDA and OpenCL kernels

The code listing in Figure 54 shows the use of the “implements” approach to express that the same functionality is available in several versions for the different devices, and allows the runtime system to decide which one is used at every task invocation.

8.3 OmpSs with support for Xilinx FPGAs and DFiant kernels

When High-Level Synthesis (HLS) is available for the target FPGA environment, as it is the case for Xilinx FPGAs, OmpSs is enabled to outline the tasks annotated as targeting the FPGA device, as C/C++ kernels that are later compiled with the HLS vendor tools (Vivado HLS for Xilinx devices). Additionally, this environment supports also to incorporate kernels written with DFiant. Figure 55 shows the toolchain for this environment.

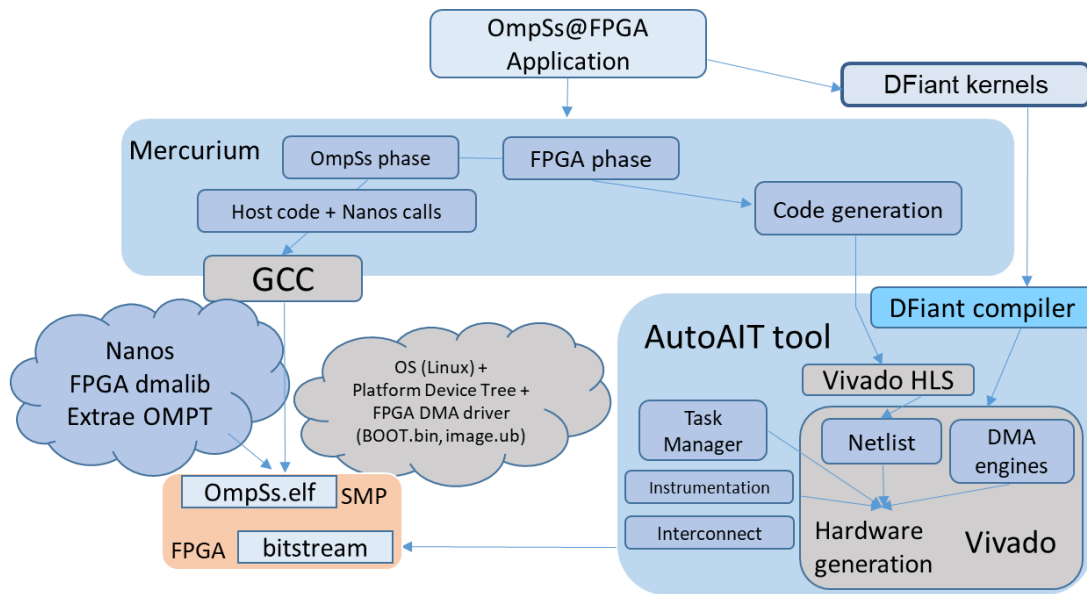


Figure 55: Diagram representation for the integration of OmpSs with CUDA and OpenCL kernels

Figure 55 shows a matrix multiplication benchmark with support for SMP and HLS tasks. It also uses the “implements” keyword to indicate to the runtime system that it can use any of the two functions to run the task functionality at any given time, provided that there are available resources. Additionally for HLS tasks, the programming model allows to express the amount of IP cores that should be generated by the compilation stage. This way, the infrastructure is already prepared to run a number of tasks in parallel on the same FPGA device (3 instances in the example).

We also provide the possibility to substitute the IP cores generated from HLS, by the kernels generated from DFiant, which are managed in the same way, integrated on the Vivado project and invoked by OmpSs at the call points.

```

// Matrix multiplication with OmpSs support for CUDA and OpenCL kernels

#pragma omp target device(smp) copy_deps
#pragma omp task in([bsize*bsize]A, [bsize*bsize]B) inout([bsize*bsize]C)
void matrixMult(REAL *C, REAL *A, REAL *B)
{
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, bsize, bsize, bsize,
                1.0f, A, bsize, B, bsize, 1.0f, C, bsize);
}

// HLS kernel task description
#pragma omp target device(fpga) copy_deps implements(matrixMult) num_instances(3)
#pragma omp task in([bsize*bsize]a, [bsize*bsize]b) inout([bsize*bsize]c)
void matrixMult_hls(REAL *c, REAL *a, REAL *b)
{
    #pragma HLS INLINE // off
    #pragma HLS array_partition variable=a cyclic factor=4
    #pragma HLS array_partition variable=b cyclic factor=BSIZE/4
    #pragma HLS array_partition variable=c cyclic factor=BSIZE/2

    for (int k = 0; k < bsize; ++k) {
        for (int i = 0; i < bsize; ++i) {
            #pragma HLS pipeline II=MBLOCK_II
            for (int j = 0; j < bsize; ++j) {
                c[i*bsize + j] += a[i*bsize + k] * b[k*bsize + j];
            }
        }
    }
}

void matmul(...)
{
    ... // Invokes block matmul and the runtime system decides which device to use for its
    execution
    matrixMult((tileC[i*nDIM+j], tileA[i*1DIM+k], tileB[k*nDIM+j],NB,NB);
}

```

Figure 56: Matrix multiplication with OmpSs support for HLS kernels

8.4 OmpSs with support for Maxeler DFEs

Figure 57 shows the integration of OmpSs with Maxeler kernels. In this approach, we have done the development at the runtime system level, allowing the programmer to use the OmpSs runtime interface to create tasks invoking the Maxeler kernels. The kernels are written in MAXJ, and compiled with the MAXJ Compiler. The resulting binary kernel file is loaded by the OmpSs runtime onto the Maxeler DFE using the SLiC interface provided by Maxeler.

The code listing in Figure 58 shows the outline of the code provided by the programmer to create, initialize and submit a blocked matrix multiplication task into the Maxeler DFE.

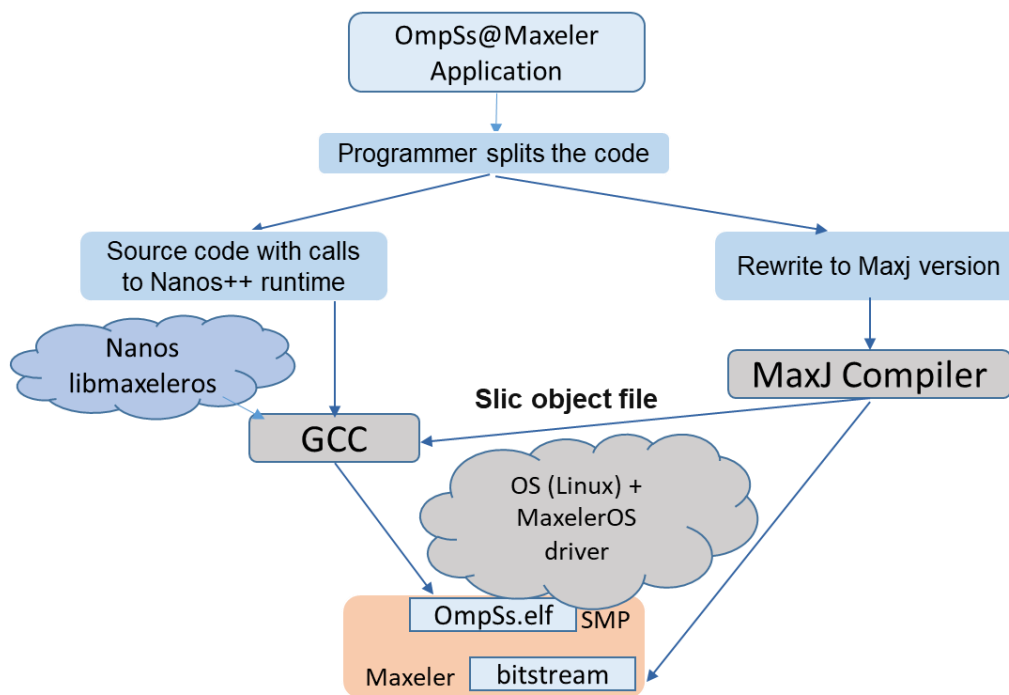


Figure 57: Diagram representation for the integration of OmpSs with Maxeler kernels


```

// Matrix multiplication with OmpSs support for Maxeler kernels

// Task creation following the target device interface of OmpSs
void matmulPanelWrapper(double *a, double *b, double *c, int panelSize)
{
    ...
    max_ol_matmulPanel_1_args.outline
        = (void (*)(void *))(void (*)(struct nanos_args_0_t
            *))*max_ol_matmulPanel_1;
    max_ol_matmulPanel_1_args.type = 1723338717;

    ...

    nanos_create_wd_compact(&nanos_wd_, &nanos_wd_const_data.base, &nanos_wd_dyn_props,
        sizeof(struct nanos_args_0_t), (void **)&ol_args, ...);

    ...
    nanos_set_translate_function(nanos_wd_,
        (nanos_translate_args_t)nanos_xlate_fun_mmttestpanelsc_0);

    ...
    nanos_submit(nanos_wd_, 3, &dependences[0], (nanos_team_t)0);
    ...
}

// Unpack the arguments of the task and perform the connections with the Maxeler kernel
void max_ol_matmulPanel_1_unpacked(double *a, double *b, double *c, int panelSize)
{
    nanos_max_queue_input("A", a, panelSize*sizeof(double));
    nanos_max_queue_input("B", b, panelSize*sizeof(double));
    nanos_max_queue_output("C", c, panelSize*sizeof(double));
}

static void max_ol_matmulPanel_1(struct nanos_args_0_t *const args)
{
    {
        max_ol_matmulPanel_1_unpacked((*args).a, (*args).b, (*args).c, (*args).panelSize);
    }
}

void __mcxx_max_register_gemm(void *p) {
    nanos_max_register_dfe( (void*)gemm_init, "TM", 1723338717);
}

```

Figure 58: Matrix multiplication with OmpSs support for Maxeler kernels

8.5 OmpSs with support for Secure SGX tasks

Figure 59 shows the integration of the secure SGX tasks with the OmpSs runtime. In this case, the programmer inserts the tasks code inside an enclave, and defines its interface using the Intel Trusted Library enclave definition language [56]. From the application side, tasks invoke the `sgx_ecall` service, passing the task parameters, and letting the Intel enclave runtime to invoke the task.

The code listing in Figure 60 shows an outline of the task code doing the offloading with OmpSs and invoking the Intel enclave services.

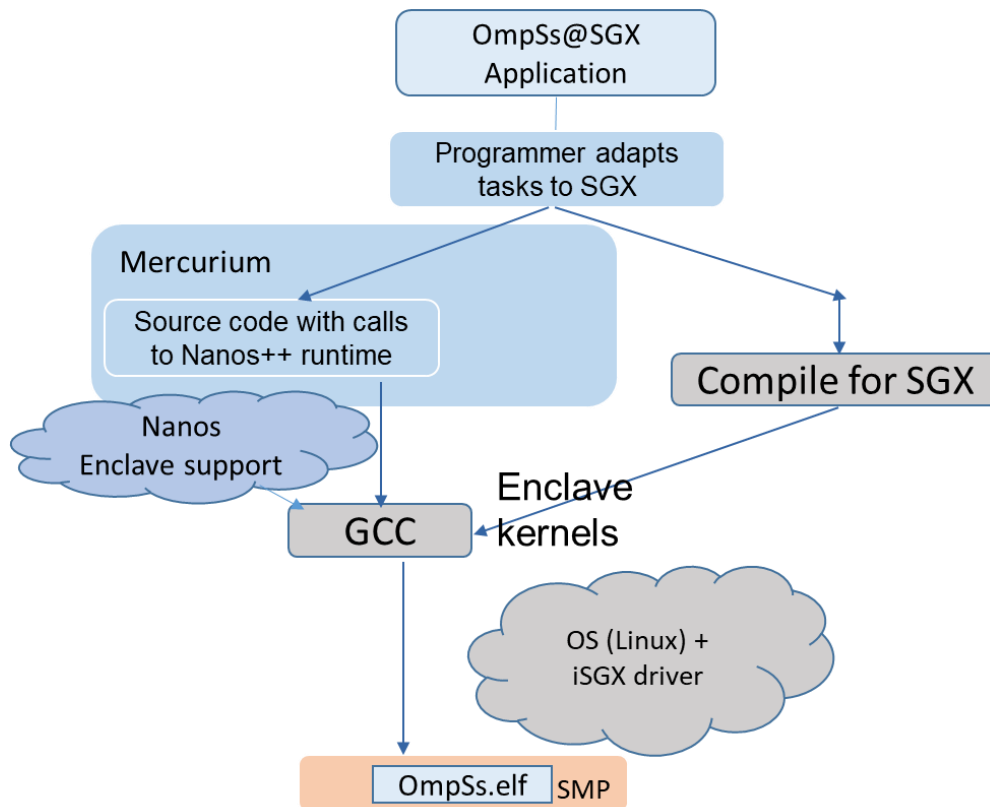


Figure 59: Diagram representation for the integration of OmpSs with secure SGX tasks

```

// Matrix multiplication with OmpSs support for secure SGX tasks

// Task entry point definition to the Enclave (Enclave/TrustedLibrary/Thread.edl)
public void ecall_matmul_u([user_check] double* a,
                           [user_check] double* b,
                           [user_check] double* c, int NB);

// Task invocations from the user application using regular OmpSs directives
int SGX_CDECL main(int argc, char *argv[])
{
    ...

    for (i = 0; i < DIM; i++) {
        for (j = 0; j < DIM; j++) {
            for (k = 0; k < DIM; k++) {
                #pragma omp task in(A[BSIZE][BSIZE], B[BSIZE][BSIZE]) inout(C[BSIZE][BSIZE]) no_copy_deps
                {
                    ecall_matmul_u(global_eid, &A[i][k], &B[k][j], &C[i][j], BSIZE);
                }
            }
        }
    }
    #pragma omp taskwait
}

// Enclave invocation
sgx_status_t ecall_matmul_u(sgx_enclave_id_t eid, double* a, double* b, double* c, int NB)
{
    sgx_status_t status;
    ms_ecall_matmul_u_t ms;
    ms.ms_a = a;
    ms.ms_b = b;
    ms.ms_c = c;
    ms.ms_NB = NB;
    status = sgx_ecall(eid, 32, &ocall_table_Enclave, &ms);
    return status;
}

// task implementation inside the Enclave
void ecall_matmul_u(double* a,
                   double* b,
                   double* c, int NB)
{
    decrypt(a, NB);
    decrypt(b, NB);
    int i, j, k, I;
    double tmp;
    for (i = 0; i < NB; i++)
    {
        I=i*NB;
        for (j = 0; j < NB; j++)
        {
            tmp=c[I+j];
            for (k = 0; k < NB; k++)
            {
                tmp+=a[I+k]*b[k*NB+j];
            }
            c[I+j]=tmp;
        }
    }
    encrypt(c, NB);
}

```

Figure 6o: Matrix multiplication with OmpSs support for secure SGX tasks

8.6 Linter tool for OmpSs

Figure 61 shows the interactions between an OmpSs running application and the Linter tool. When the application creates and manages tasks, those and their data dependence information are communicated to the Linter tool. The Linter has also access to the information expressed on the directives regarding data hints, and to the debugging information generated by the compiler. In this way, the Linter tool is able to detect data races, and missing directive hints, and report them to the user. The development of the Linter tool has been shown in Deliverable 3.3 [57].

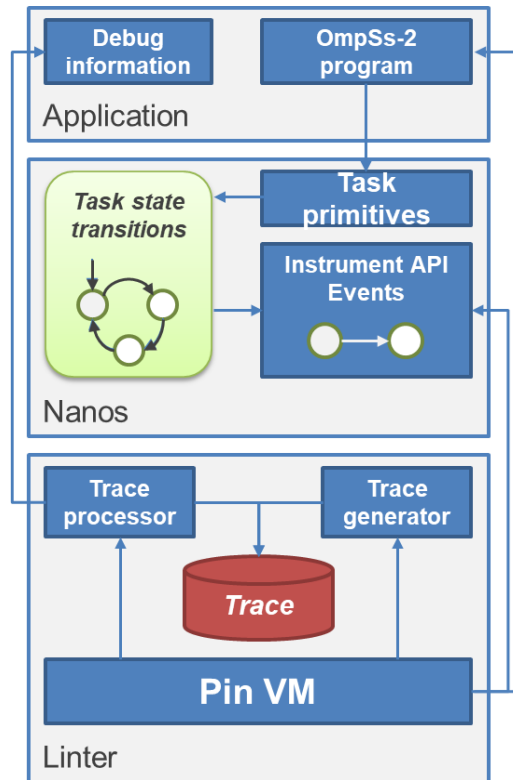


Figure 61: OmpSs Application and Linter tool execution diagram

8.7 IDE plugin for OmpSs

Figure 62 shows an Eclipse Che workspace with the support for the OmpSs@FPGA compilation environment. It allows to manage projects for FPGA acceleration and build the binary for the host and the bitstream for the FPGA automatically.

We have also developed the Eclipse plugins that provide hints for the programmer for OmpSs and OpenMP. Figure 63 shows the IDE plugin displaying information about the possibilities available at the writing point for task clauses, and among them the data-directionality hints (in, inout), and an explanation of their meaning. The development of the Eclipse plugin is presented in Deliverable 4.4 [58].

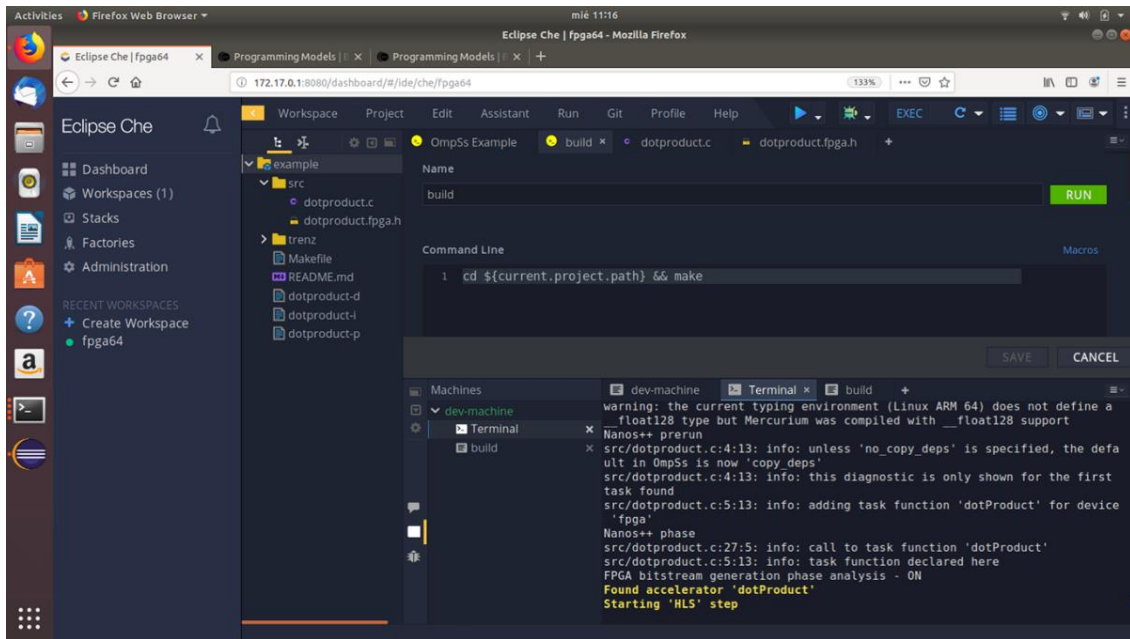


Figure 62: Screenshot of an Eclipse Che workspace with the compilation of the dot-product sample

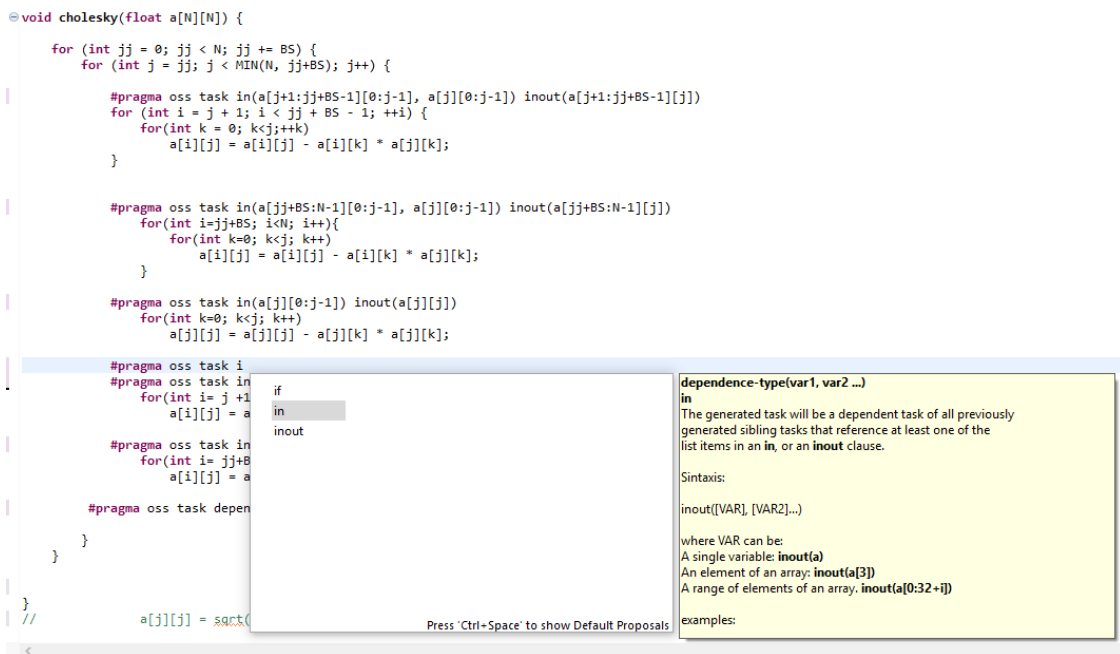


Figure 63: Eclipse plugin showing the hints offered to the programmer about task data-directionality hints

8.8 RECS Master and Slurm

The Slurm Workload Manager [14] has been integrated with the RECS_Master node management service. With this integration, users can allocate nodes in the cluster by using Slurm scripts. Figure 64 shows the batch job script description that the user submits to the queueing system. In this case, the constraint used is that the nodes should be of the ARM architecture, with big.LITTLE cores, and a GPU. Slurm allocates the nodes, and the job is submitted to them for execution.

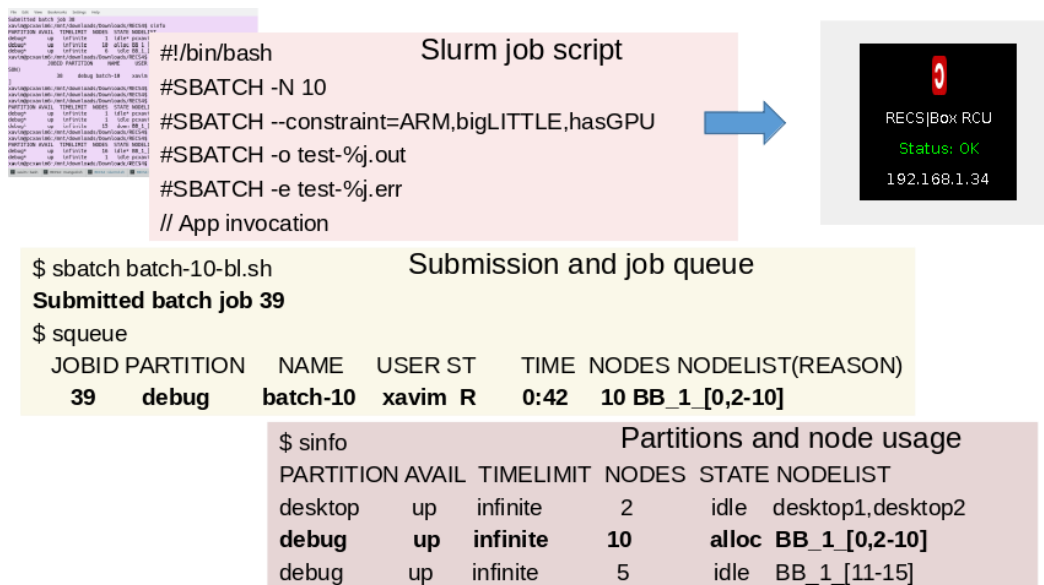


Figure 64: Integration of Slurm with the RECS Master management environment

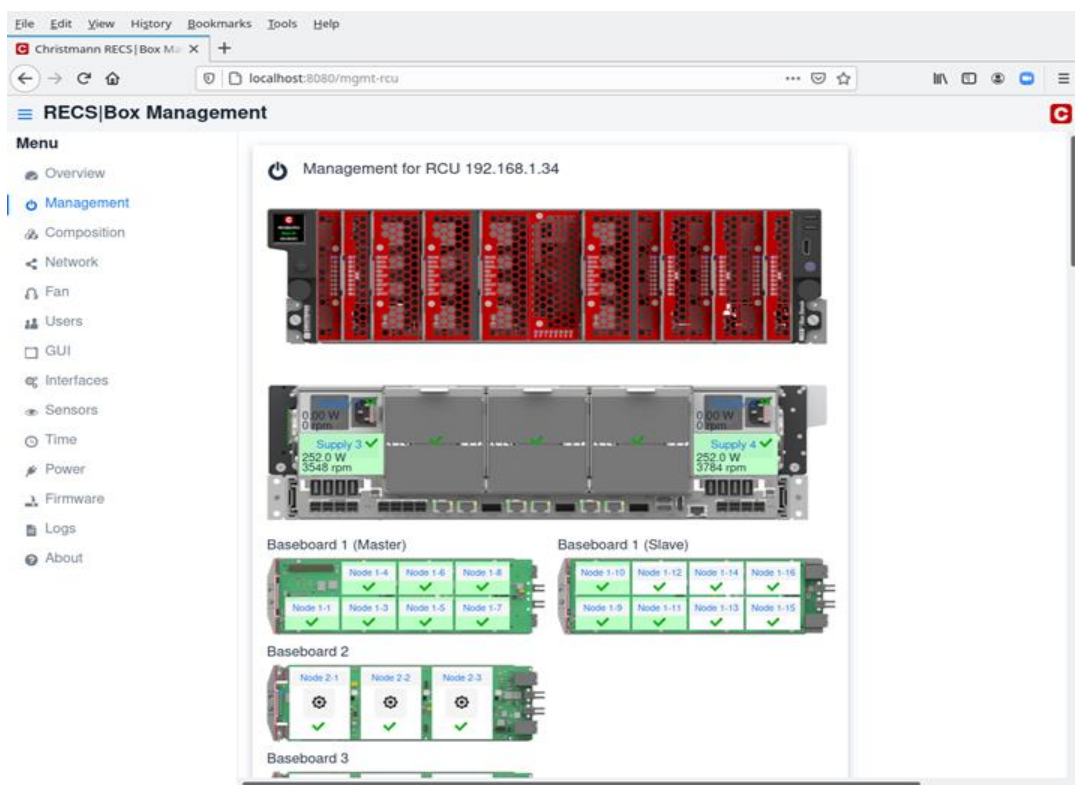


Figure 65: Display of nodes allocated using the Slurm integration on the Web-GUI of RECS_Master

Figure 65 shows the display of the user interface of RECS_Master, indicating that the 10 nodes are up and running while the job is running. Upon completion, Slurm will put the nodes back in the off state. The Slurm and RECS_Master integration is shown in Deliverable 3.4 [59].

9 Conclusion

This document presents the final implemented and optimised use cases, as well as benchmarks comparing the original with the optimised versions.

We were able to show (partially enormous) improvements in all targeted disciplines: energy efficiency, MTBF, code base security, designer productivity, TCO/costs and latency – as detailed in the introduction. Four of five use cases were improved with a focus on total speedup and energy efficiency, making use of the various LEGaTO technologies, reaching up to the impressive factor of 2503.6x speedup in the Heath Care use case.

Metric	Target	Achieved
Energy Efficiency	10x increase	Smart Home: 12x , c.f. Chapter 3.4 Smart City: 35x - 95x , c.f. Chapter 4.4 Infection Research: 8.35x - 7708.8x , c.f. Chapter 5.3 Machine Learning: 11.4x , c.f. Chapter 6.5
MTBF, application reliability	5x increase	Smart City: 15.23x , c.f. Chapter 4.6
Code Base Security	10x increase	Infection Research: 7.7x , c.f. Chapter 5.3
Designer Productivity	5x increase	DFiant: 50% - 70% less lines of code , c.f. D4.4 Chapter 4.1.1.5 MaxJ: 4x improvement of productivity [60], also c.f. D4.4 Chapter 3.1 Additional improvements are expected through the usage of OmpSs and the developed Eclipse IDE
TCO & Costs	n/a	Smart Home: 81.6% TCO improvement , c.f. D5.4 Chapter 4 Smart City: 70% TCO improvement , c.f. D5.4 Chapter 5 Infection Research: 91.8% - 98.8% TCO improvement , c.f. D5.4 Chapter 6 Machine Learning: 43.8% TCO improvement , c.f. D5.4 Chapter 7 Secure IoT Gateway: 40% TCO improvement , c.f. D5.4 Chapter 8
Latency	n/a	Machine Learning: 11x , c.f. Chapter 6.5

Table 17: Overview of evaluated and optimised metrics

The Smart Home use case was able to achieve the goal of running on the very efficient LEGaTO-developed edge server prototype, increasing the energy efficiency by 12x. The Smart City use case showed an energy efficiency increase of 95x running on an Nvidia Xavier microserver at the cost of an 11x slowdown, compared to a BSC Marenosturm4 node. The Infection Research use case was able to massively speed-up not only one, but four different application parts, resulting in a speed-up of 10x, 40x, 544x and the mentioned 2503.6x. This opens up a whole new world of statistical analytics that was impossible to do before. The Machine Learning use case developed the new deep learning optimisation tool EmbeDL, showing an average speed-up of 4.3x and energy efficiency

increase of 6.3x. The Secure IoT Gateway was developed, easing the usage of IoT security through a web interface and deployment processes, and successfully integrated into two locations of the Smart Home use case.

The usage of the diverse LEGaTO tools was eased as OmpSs was chosen as the main programming model for most use cases. Therefore, OmpSs was extended to make use of additional compilers and runtimes as described in Chapter 8, so the use cases didn't have to implement all of them in a different way.

All these developments and optimisations prove that LEGaTO was very successful in the overall project integration, bringing the developed hardware, middleware and whole optimisation toolchains to life and use them in completely different use cases. As some of the use cases are research-oriented, this will not only open up new research fields but also encourage similar research in the area of energy efficiency in Europe to use matured software stacks like OmpSs. Other use cases open up new fields of engagement and research because of disruptive improvements like the Infection Research use case.

Further fields of development and research have partially been discussed in the individual sections, but are summarised here. The Smart Home use case could stabilize and mature the actual developments, design an attractive overall smart mirror product and sell the solution – besides further research in human interaction and smart living. The Smart City needs to further research on the scaling of the very efficient implementation as the total performance is hardly comparable to today's classical solution. The Infection Research use case wants to further research on the developed and optimised application parts, publish the research, but also bundle the application parts for easy usage for external people. This would open up a new world of statistical biomarker analysis for the whole medical community. The Machine Learning use case has a clear plan on how to continue the development in the EU funded follow up project VEDLIoT, and in parallel is marketing the solution via the spin-off EmbeDL. The development of the Secure IoT Gateway will also be continued in the VEDLIoT project, adding new features like WiFi and LoRa support and increasing the TRL level from 6 to 7, targeting a market launch within the next two years.

10 References

- [1] H2020 LEGaTO Project, "Deliverable 2.1: Architecture Definition and Evaluation Plan for Legato's Hardware, Toolbox and Applications," 2018.
- [2] H2020 Project M2DC, "Deliverable 5.2: First report on development and optimization of use-cases," 2019.
- [3] H2020 Project M2DC, "Deliverable 5.4 - Report on evaluation of efficiency and TCO improvements of use-cases," 2020.
- [4] M. Teeuw, "MagicMirror² GitHub page," [Online]. Available: <https://github.com/MichMich/MagicMirror>. [Accessed 6 11 2020].
- [5] LEGaTO Project, "LEGaTO smart mirror module GitHub page," [Online]. Available: <https://github.com/LEGaTO-SmartMirror?tab=repositories>. [Accessed 6 11 2020].
- [6] European Environment Agency, "Air quality in Europe, 2018 Report, EEA Report No. 12/2018," European Union, 2018.
- [7] National Center for Atmospheric Research (NCAR), "The weather research and forecasting model," [Online]. Available: <https://www.mmm.ucar.edu/weather-research-and-forecasting-model>. [Accessed 17 7 2019].
- [8] CASE Department – BSC, "Alya – High Performance Computational Mechanic Simulator," [Online]. Available: <https://www.bsc.es/research-development/research-areas/engineering-simulations/alya-high-performance-computational>. [Accessed 19 7 2019].
- [9] AXIOM Project, "The AXIOM Board has arrived!," [Online]. Available: <http://www.axiom-project.eu/2017/02/the-axiom-board-has-arrived>. [Accessed 2019 7 17].
- [10] Cavium, "Mont-Blanc project selects Cavium's ThunderX2™ processor for its new ARM-based HPC platform," [Online]. Available: <https://www.cavium.com/News-Release-Mont-Blanc-project-selects-Caviums-ThunderX2-processor-for-its-new-ARM-based-HPC-platform.html>. [Accessed 19 7 2019].
- [11] Alpha Data, "ADM-PCIE-7V3 – High Performance Computing," [Online]. Available: <https://www.alpha-data.com/dcp/products.php?product=adm-pcie-7v3>. [Accessed 19 7 2019].
- [12] R. A. a. A. G. Oyarzun, "Portable implementation model for CFD simulations. Application to hybrid CPU/GPU supercomputers," *International Journal of Computational Fluid Dynamics*, pp. 396-411, 2017.

- [13] N.-Y. J. Hosseinabady M., "A Streaming Dataflow Engine for Sparse Matrix-Vector Multiplication using High-Level Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.
- [14] M. J. M. G. A.B. Yoo, "SLURM: Simple Linux Utility for Resource Management," in *Proceedings of the 9th Job Scheduling Strategies for Parallel Processing*, Berlin, 2003.
- [15] N. Corporation. [Online]. Available: <https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3231/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/AppendixTegraStats.html> .
- [16] H2020 Project LEGaTO, D3.3: Final Release of the Task-Based Runtime, 2020.
- [17] K. Parasyris, K. Keller, L. Bautista-Gomez and O. Unsal, "Checkpoint Restart Support for Heterogeneous HPC Applications," *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pp. pp. 242-251.
- [18] R. Pepperkok and J. Ellenberg, "High-throughput fluorescence microscopy for systems biology," in *Nature Reviews Molecular Cell Biology* 7, 690-696, 2006.
- [19] A.-J. WAN, K. Wang, H.-C. ZHANG, H.-L. LI and D.-N. WANG, "Modercarbohydrate microarray biochip technologies," in *Chinese Journal of Analytical Chemistry* 40(11): 1780–1788, 2012.
- [20] W. W. Soon, M. Hariharan and M. P. Snyder, "High-throughput sequencing for biology and medicine," in *Molecular systems biology* 9, page 640, 2013.
- [21] Programming MPC Systems, White Paper, 2013.
- [22] G. M. Q. F. T. W. T. C. W. M. W. Y. Q. & L. T. Y. Ke, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Advances in Neural Information Processing Systems*, pp. 3147-3155, 2017.
- [23] T. S. S. Y. T. O. T. a. K. M. Akiba, "Optuna: A next-generation hyperparameter optimization framework," *KDD*, 2019.
- [24] R. Tibshirani, "Regression Shrinkage and Selection Via the Lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, pp. 267-288, 1996.
- [25] R. A. J. . In-Kwon Yeo, "A new family of power transformations to improve normality or symmetry," *Biometrika*, pp. 954-959, 2000.
- [26] Lower Saxony Ministry for Science and Culture in Germany, *Den Weg zu individuellen Impfungen finden*, <https://www.mwk.niedersachsen.de/downloads/145045>: Big data in the future life sciences, 2019.
- [27] Niedersächsisches Ministerium für Wissenschaft und Kultur, "Big Data in den Lebenswissenschaften der Zukunft - Chancen datenintensiver Forschung und personalisierter Medizin," 29 06 2018. [Online]. Available:

https://www.mwk.niedersachsen.de/startseite/forschung/forschungsforderung/ausschreibungen_programme_forderungen/big-data-in-den-lebenswissenschaften-der-zukunft--166104.html. [Accessed 26 11 2020].

- [28] R. a. N. S. Suthar, "Bacterial Infections of the Central Nervous System.," *Indian J Pediatr*, pp. 60-69, 2019.
- [29] "musl libc project website," [Online]. Available: <https://musl.libc.org/>. [Accessed 26 11 2020].
- [30] "TensorFlow Project Website," [Online]. Available: <https://www.tensorflow.org/>. [Accessed 27 11 2020].
- [31] "PyTorch Project Website," [Online]. Available: <https://pytorch.org/>. [Accessed 27 11 2020].
- [32] "Caffe Deep Learning Framework Project Website," [Online]. Available: <https://caffe.berkeleyvision.org/>. [Accessed 27 11 2020].
- [33] "ONNX - Open Neural Network Exchange Project Website," [Online]. Available: <https://onnx.ai/>. [Accessed 27 11 2020].
- [34] M. Möller, "Open Hardware Monitor Project Website," [Online]. Available: <https://openhwaremonitor.org/>. [Accessed 27 11 2020].
- [35] Nvidia, "Nvidia Jetson AGX Xavier - Thermal Design Guide," Nvidia, https://static5.arrow.com/pdfs/2018/12/12/22/1/565659/nvda_/manual/jetson_agx_xavier_thermal_design_guide_v1.0.pdf, 2018.
- [36] zedboard.org, "Zynq™ Evaluation and Development Hardware User's Guide," 27 01 2014. [Online]. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf. [Accessed 26 11 2020].
- [37] A. Zisserman and K. Simonyan, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, <https://arxiv.org/abs/1409.1556>, 2015.
- [38] K. He, X. Zhang, S. Ren and J. Sun, *Deep Residual Learning for Image Recognition*, <https://arxiv.org/abs/1512.03385>, 2015.
- [39] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2012.
- [40] J. Redmon and A. Farhadi, *YOLOv3: An Incremental Improvement*, <https://arxiv.org/pdf/1804.02767.pdf>: University of Washington, 2018.
- [41] Lin, Tsung-Yi et al., *Microsoft COCO: Common Objects in Context*, ECCV 2014: Computer Vision – ECCV 2014 pp 740-755: Springer, 2014.
- [42] Deciso B.V., "OPNsense Project Website," [Online]. Available: <https://opnsense.org/>. [Accessed 27 11 2020].

- [43] "OpenWrt Project Website," [Online]. Available: <https://openwrt.org/>. [Accessed 27 11 2020].
- [44] "Lua Project Website," [Online]. Available: <https://www.lua.org/>. [Accessed 27 11 2020].
- [45] "Nuxt.js - The Intuitive Vue Framework," [Online]. Available: <https://nuxtjs.org/>.
- [46] "Bulma: Free, open source, and modern CSS framework based on Flexbox," [Online]. Available: <https://bulma.io/>. [Accessed 5 11 2020].
- [47] "Prisma - Database tools for modern application development," [Online]. Available: <https://www.prisma.io/>. [Accessed 5 11 2020].
- [48] "Express - Node.js web application framework," OpenJS Foundation, [Online]. Available: <https://expressjs.com/>. [Accessed 5 11 2020].
- [49] "GraphQL | A query language for your API," Linux Foundation, [Online]. Available: <https://graphql.org/>. [Accessed 5 11 2020].
- [50] "KogniHome - Technikunterstütztes Wohnen für Menschen e.V.," KogniHome Technikunterstütztes Wohnen für Menschen e. V., [Online]. Available: <https://www.kognihome.de/>. [Accessed 9 11 2020].
- [51] "MQTT - The Standard for IoT Messaging," OASIS, [Online]. Available: <https://mqtt.org/>. [Accessed 9 11 2020].
- [52] "Research Institute for Cognition and Robotics | cor-lab.de," Universität Bielefeld - CoR-Lab, [Online]. Available: <http://docs.cor-lab.de/rsb-manual/o.g/html/concepts.html>. [Accessed 9 11 2020].
- [53] J. Schwarz and K. Schwarz, "YouTube," 13 06 2018. [Online]. Available: <https://www.youtube.com/watch?v=LXb3EKWsInQ>. [Accessed 09 11 2020].
- [54] "Wikipedia: Matrix Multiplication," [Online]. Available: https://en.wikipedia.org/wiki/Matrix_Multiplication. [Accessed 24 11 2020].
- [55] Nvidia Corporation, "CUDA Toolkit Documentation v11.1.1," [Online]. Available: <https://docs.nvidia.com/cuda/index.html>. [Accessed 24 11 2020].
- [56] Intel Corporation, "Intel Software Guard Extensions (Intel SGX)," [Online]. Available: https://download.01.org/intel-sgx/sgx-linux/2.12/Intel_SGX_Developer_Guide.pdf. [Accessed 24 11 2020].
- [57] H2020 Project LEGaTO, *Deliverable 3.3: Final Release of the Task-Based Runtime*, May 2020.
- [58] H2020 Project LEGaTO, *Deliverable 4.4: Report on Energy-Efficiency Evaluations and Optimizations for Energy-Efficient, Secure, Resilient Task-Based Programming Model and Compiler Extensions*, November 2020.

- [59] H2020 Project LEGaTO, *Deliverable 3.4: Report on evaluation and optimizations in the runtime stack*, November 2020.
- [60] Voss, Nils et. al., "Rapid Development of gzip with MaxJ," *Applied Reconfigurable Computing (ARC)*, 2017.
- [61] V. B. Patel and V. R. Preedy, "General Methods in Biomarker Research and their Applications," 2015.
- [62] C. Endrullat, J. Glökler, P. Franke and M. Frohme, "Standardization and quality management in next-generation sequencing," *Applied and Translational Genomics*, 2016.
- [63] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *ICLR 2015*, San Diego, CA, USA, 2015.
- [64] C. Szegedy, W. Liu, J. Yangqing, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [65] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *CoRR*, 2015.
- [66] J. Redmon, "YOLO: Real-Time Object Detection," [Online]. Available: <https://pjreddie.com/darknet/yolo/>.
- [67] NVIDIA, "NVIDIA Tensor Cores product page," [Online]. Available: <https://www.nvidia.com/en-us/data-center/tensorcore/>. [Accessed 31 07 2019].
- [68] NVIDIA, "NVIDIA TX2 product page," [Online]. Available: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/>. [Accessed 31 07 2019].
- [69] NVIDIA, "NVIDIA Xavier product page," [Online]. Available: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/>. [Accessed 31 07 2019].
- [70] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175, 2006.
- [71] "OpenWrt Project," [Online]. Available: <https://openwrt.org/>. [Accessed 5 11 2020].
- [72] "OPNsense® a true open source security platform and more," Deciso B.V, [Online]. Available: <https://opnsense.org/>. [Accessed 5 11 2020].