http://www.everest-h2020.eu

dEsign enVironmEnt foR Extreme-Scale big data analyTics on heterogeneous platforms

D2.5 — Refined Definition of Language Requirements

The EVEREST project has received funding from the European Union's Horizon 2020 Research & Innovation programme under grant agreement No 957269

Project Summary Information

Project Title	dEsign enVironmEnt foR Extreme-Scale big data analyTics on heterogeneous platforms
Project Acronym	EVEREST
Project No.	957269
Start Date	01/10/2020
Project Duration	42 months
Project Website	http://www.everest-h2020.eu

Copyright

Num.	Partner Name	Short Name	Country
1 (Coord.)	IBM RESEARCH GMBH	IBM	CH
2	POLITECNICO DI MILANO	PDM	IT
3	UNIVERSITÀ DELLA SVIZZERA ITALIANA	USI	CH
4	TECHNISCHE UNIVERSITAET DRESDEN	TUD	DE
5	Centro Internazionale in Monitoraggio Ambientale - Fondazione CIMA	CIMA	IT
6	IT4Innovations, VSB – Technical University of Ostrava	IT4I	CZ
7	VIRTUAL OPEN SYSTEMS SAS	VOS	FR
8	DUFERCO ENERGIA SPA	DUF	IT
9	NUMTECH	NUM	FR
10	SYGIC AS	SYG	SK

Project Coordinator: Christoph Hagleitner – IBM Research – Zurich Research Laboratory

Scientific Coordinator: Christian Pilato – Politecnico di Milano

Disclaimer

Deliverable Information

Work-package	WP2
Deliverable No.	D2.5
Deliverable Title	Refined Definition of Language Requirements
Lead Beneficiary	TUD
Type of Deliverable	Report
Dissemination Level	Public
Due Date	31/01/2023

Document Information

Delivery Date	31/03/2023
No. pages	34
Version | Status	0.4 | Final
Responsible Person	Jeronimo Castrillon (TUD)
Authors	Jeronimo Castrillon (TUD), Felix Suchert (TUD), Jakub Beránek (IT4I), Karl Friebel (TUD), Martin Šurkovský (IT4I), Serena Curzel (PDM), Christian Pilato (PDM), Dionysios Diamantopoulos (IBM), Burkhard Ringlein (IBM)
Internal Reviewer	Francesco Regazzoni (USI)

The list of authors reflects the major contributors to the activity described in the document. All EVEREST partners have agreed to the full publication of this document. The list of authors does not imply any claim of ownership on the Intellectual Properties described in this document.

Revision History

Date	Ver.	Author(s)	Summary of main changes
12.10.2022	0.1	Jeronimo Castrillon (TUD)	Created using some basic information from D2.2.
06.12.2022	0.2	Jeronimo Castrillon (TUD)	Ported from D2.2 and distributed tasks.
03.01.2023	0.3	Jeronimo Castrillon (TUD)	Adapted tables.
30.01.2023	0.4	Jeronimo Castrillon (TUD) and others	Finished updates to Sections 3–7.

Quality Control

Approved by Internal Reviewer	23/03/2023
Approved by WP Leader	31/03/2023
Approved by Scientific Coordinator	31/03/2023
Approved by Project Coordinator	31/03/2023

Table of Contents

1 Executive Summary

The EVEREST project aims to design a platform for implementing big data applications with both high performance and edge workloads following a data-driven model. With the goal of designing the programming interface for this envisioned platform, we studied the use cases of the EVEREST project. This document provides an updated report on the results of this study and lists requirements on languages and tooling to be developed for the EVEREST programming framework. Together with the application requirements reported in Deliverable D2.1 and Deliverable D2.4 and the data requirements formulated in Deliverable D2.3 and Deliverable D2.6, they define the work on the EVEREST design environment to be done in work packages 3-6. Therefore, these deliverables are closely linked and were carefully checked for consistency. Despite the many links between, e.g., the language and application requirements, we attempted to make the three documents self-contained and easy to read. Therefore, some basic requirements are stated in several deliverables. Cross-linking all of them would make the individual documents unreadable.

We observe two different major workload types in the use cases. The first type is characterized by single-location heavy computational workload (e.g., weather simulations, or machine learning). The second type corresponds to computation distributed across loosely coupled systems, like data acquisition tasks. The highest potential gain achievable by specialized language support is exhibited by the heavy computational workloads that directly profit from the novel heterogeneous node architecture of the EVEREST platform. We thus propose a custom tool flow with tailor-made domain-specific abstractions coupled with runtime components which enables us to achieve high interoperability and retargetability at a low cost to users of existing code bases. In addition, to unlocking the potential of EVEREST nodes, we consider language support for coordination tasks to better support the second type of workload. We describe how the proposed language support and associated tooling integrates with existing code bases and development environment. To accomplish this, we derive requirements on the different tools and system software of the EVEREST programming framework, including compilers, runtime auto-tuner, runtime system and high-level synthesis tools.

This deliverable represents an update to the initial requirement analysis reported in Deliverable D2.2. Major changes are:

- Section 3: Minor changes to reflect new information on the use cases. Further details are provided in Deliverable D2.4.
- Section 4: More information about the traffic-use case and its components. Better role definition for the programming language and the requirements for task management in HyperQueue. This resulted in minor modifications to the previous requirements and three new requirements on the language and compiler (REQ3.12 - REQ3.14) in Table 5.
- Section 5: With the more precise definition of the traffic use case, the problem description for multiple parallel inference models is clearly stated. The interchange format for Machine Learning (ML) models has been also fixed and extensions needed to represent quantized models were identified. Moreover, the radiation module in weather simulations is better understood and thus more precise requirements were derived for the Domain-Specific Language (DSL) and associated high-level compiler. We added the Helmholtz operator from fluid dynamics to steer the requirements on the toolflows with a more self-contained yet challenging numerical kernel.
- Section 6: as a consequence of data requirements being better understood, this section now more clearly specifies how High-Level Synthesis (HLS) tools need to account for large data sets and module interfacing, and what new tools on top of HLS flows need to support to feed data to hardware accelerators. This changes led to minor modifications to the previous requirements in Table 6. While the bandwidth between the HPC compute nodes and on the bus-interfaces between the CPU and the FPGA-accelerators within the compute nodes is important, the ability to coherently access address memory across the CPU and accelerator complex was not identified as an important element during the analysis of the use-cases. Furthermore, the OpenCAPI interface, which we were planning to use to link the FPGAs to the CPUs did not achieve the wide-spread adoption we were expecting at the time of writing the proposal. Therefore, the EVEREST consortium decided to implement the FPGA-accelerated HPC system that will be used

to demonstrate acceleration of the weather codes based on standard PCIe-attached FPGA-accelerator cards (Xilinx Alveo). Our proactive decision at the early stages of the project was confirmed recently, when the OpenCAPI consortium agreed to transfer all OpenCAPI consortium assets to the CXL Consortium [6]. With the new focus on PCIe-attached FPGA cards, EVEREST is well positioned to adopt CXL-attached devices with support for coherency once they become available.

- Extra-functional information: Deliverable D2.2 included a section (Section 7) which discussed possible ways to capture extra functional constraints. More concretely, we considered how to represent (1) timing constraints for real time execution, (2) energy efficiency, and (3) data security. Real-time execution was not relevant to the use cases, so it is not included in this document. The discussion about energy efficiency in this requirements document was ill-placed. Execution metrics, in general, are discussed in Deliverable D2.4. Similarly, data considerations are discussed in Deliverable D2.6.

1.1 Structure of this Document

Section 2 first introduces the overall aim of the EVEREST project. Section 3 breaks down the use cases of the project to identify functionality that could profit from domain-specific optimizations and hence should be supported by the EVEREST programming framework. Part of this analysis looks for libraries and code sections that are shared between the use cases. In Section 4 and Section 5, functional requirements for the language abstractions to be developed are analyzed and discussed. Section 6 refers to the requirements considerations related to the FPGA experimental research platforms of EVEREST. Section 7 provides a detailed requirement analysis on the different components of the EVEREST programming framework. Finally, Section 8 concludes this report.

1.2 Related Documents

This report is closely related to:

- D2.1 - Definition of the Application Uses Cases,
- D2.3 - Definition of data requirements,
- D4.1 - Definition of the compilation framework (M9),
- D2.4 - Refined definition of application uses-cases (M24),
- D2.6 – Refined definition of data requirements (M24).

2 Introduction

The EVEREST project will implement a platform for heterogeneous, distributed, scalable, and secure High-Performance Big Data Analytics (HPDA). The design of a programming framework and the underlying programming abstractions take a key role in this undertaking. Providing designers with widely platform-agnostic development tools that can perform meaningful optimizations on the code poses a unique challenge. These tools should seamlessly integrate into current development flows, requiring minor changes to established practices.

The development of the platform and programming framework is driven by three industry-relevant use cases, namely **renewable-energy prediction**, **air-quality monitoring**, and **traffic modeling**. Apart from the high societal relevance of these use cases, they are excellent representatives of HPDA, combining challenging high-performance computing, machine learning (ML) modeling, and state of the art algorithms for decision making. Given the use case heterogeneity, established programming practices differ across the different domains. This makes it even more challenging to design and implement a seamless programming framework.

This report summarizes the requirements for language abstractions and the programming framework as a whole. We start by introducing the use cases in more detail, extracting important underlying computational patterns that can profit from language and compiler support. One such pattern are HPC and ML kernels, common to the different use cases as well as the coordination and workflow aspects of the applications. These aspects will steer the development of the big data framework and associated language abstractions by means of dataflow models. We also describe extra-functional constraints that have to be respected for the different use cases. Finally, we list identified requirements for the different components of the use cases and the EVEREST programming framework.

3 Use Case Analysis

The EVEREST platform will support applications that process large amounts of data in a distributed setting. In order to better understand the requirements and find optimization potential in such applications, we first analyze the use cases of the project to identify common bottlenecks and aspects that can profit from language support.

This section briefly discusses the three use cases, highlighting aspects relevant for the design of the programming framework. A more detailed presentation of the use cases can be found in Deliverable D2.1 and Deliverable D2.4 . All use cases are at different levels of maturity, as is explained in detail in the aforementioned documents.

3.1 Air Quality Monitoring

Industrial plants that emit pollution are naturally subject to strict regulation defining acceptable levels of air quality which must be met. But depending on the weather situation, the emission dissipation greatly varies. This use case intends to provide local monitoring of the air quality and weather on site to help regulating the pollution produced by a plant accordingly. Weather data is analyzed and used to produce a local weather forecast for a 10km radius around pollution sources. The resulting information will then be combined with machine learning approaches to assist in deciding whether or not to postpone emission-heavy activities at the industrial site.

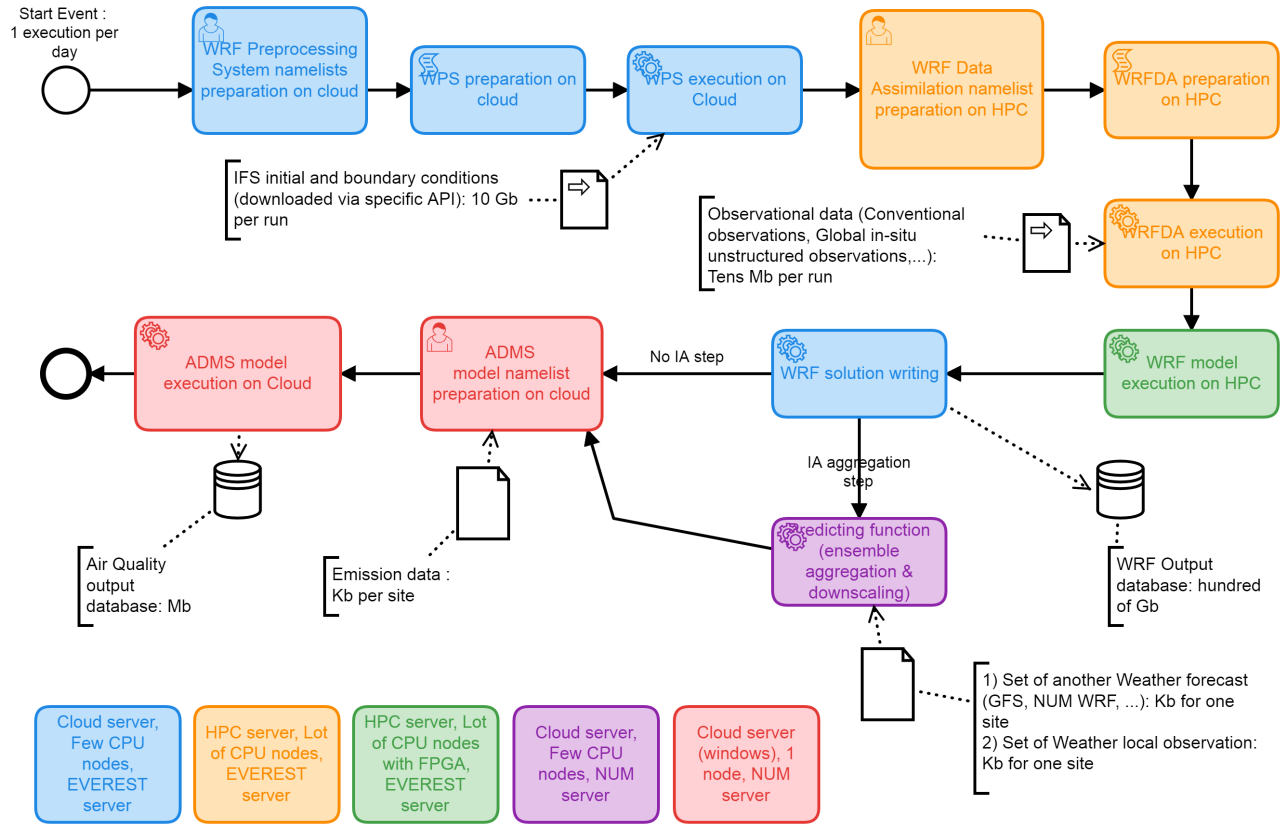

Figure 1 – Air quality monitoring use case flow

The overall structure of this use case is shown in Figure 1. It combines different functions with different computational requirements and computational patterns, executed across multiple sites and systems. The largest and computationally most intense aspect of this application is the generation and simulation of the weather model using the WRF model. It consists of many different kernels computing and simulating the differ-

ent facets that influence weather phenomena such as cloud movement and radiation. Due to the impact these kernels have on the computation time of the application, the WRF model is a key component for acceleration. Given the experience of the partners in DSLs for Computational Fluid Dynamics (CFD), these kernels will serve as initial target for DSL design. From appropriate abstractions, compilers can be designed/extended in WP4 that lower the code to multi-core computing nodes with and without FPGA acceleration.

3.2 Renewable-Energy Prediction

To better harvest the potential of Renewable Energy sources, this application will provide a predictive model to forecast upcoming weather events that may influence the energy production from renewable sources. It will analyze real-time weather data and generate a high-resolution weather model that can produce highly localized weather forecasts hourly or sub-hourly. Artificial intelligence methods will then be used on the output generated by the weather model to estimate possible productions by renewable energy sources.

Like the Air Quality use case, this application is mainly built around a weather model to generate forecasts. Kernel optimization for the model will thus benefit this use case as well.

3.3 Traffic Modelling

The main goal of this use case is to optimize traffic flows within cities to reduce congestion and travel times, which in turn can help reducing the pollution caused by traffic. Based on both historical and real-time traffic data, a traffic simulation is run in conjunction with a prediction model to allow the forecasting of high-congestion scenarios and route the traffic accordingly when routes are requested.

This use case consists of several interconnected workflows that together form a larger application, as shown in Figure 2. The workflows such as Map Matching, Traffic simulation, Traffic prediction inference and Intelligent routing contain kernels that can profit from performance improvements brought by the use of custom domain-specific abstractions. By using higher-level abstractions, it will be possible for these kernels to transparently leverage the compute efficiency of accelerators implemented on the reconfigurable fabric of the EVEREST nodes. Apart from kernels, this use case requires orchestration of the individual workflows, which are either streaming-based or pure batch processing, especially the Traffic Simulator.

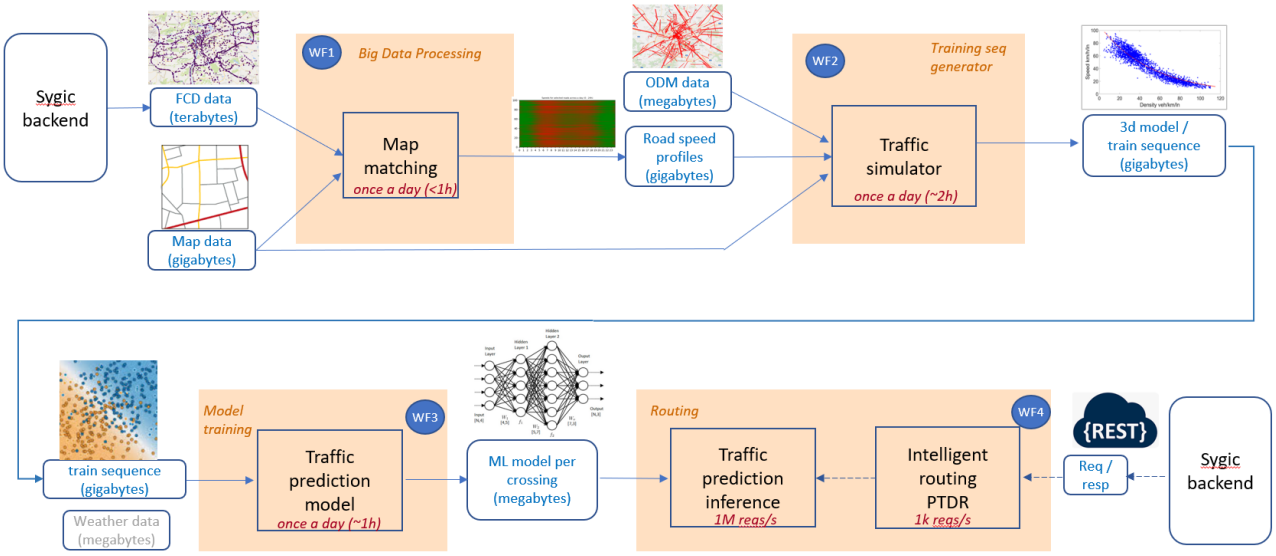

Figure 2 – Outline of the traffic simulation use case

In the next sections we will discuss the specific challenges posed by the use cases. In order to make solutions to these challenges widely applicable within the EVEREST framework, we will abstract from the specific issues, focusing on generalizable solutions.

4 Workflow Distribution and Parallelization

As outlined in Section 3, the different use cases of the EVEREST project are large-scale applications that would benefit from distributed and parallelized execution on multiple levels. Within the use cases, WRF is a traditional High-Performance Computing (HPC) application with a well-understood distributed execution model. This section thus focuses on parallelization and large-scale distribution for other workflows, namely, the traffic simulator, map matching and inference.

4.1 Problem Description

The traffic use-case consists of several components, which have different requirements and potential for being accelerated by distributed computation.

- **Traffic Simulator** The traffic simulator component is used to generate large amounts of data by executing simulations of cars travelling within various cities. This data is then used as an input dataset for training traffic modelling neural networks. Due to the nature of the service, fast data processing is required to deliver timely results which can be used to react to current circumstances in the field.

  On a coarse-grained, workflow orchestration level, it will be required to execute many individual instances of the traffic simulator, with various parameters and cluster configurations (varying node counts, leveraging only CPU nodes or also FPGA-accelerated nodes etc.).

  Within each traffic simulator workflow, computation should also be distributed, on a more fine-grained level. Within each iteration of a batch of cars whose movement should be simulated, it is required to compute a set of alternative potential routes for each car and then perform a Monte Carlo simulation to evaluate these routes. Since both alternative route finding and the probabilistic Monte Carlo simulations can be computed in an embarrassingly parallel fashion for each car, they will be computed in parallel on multiple nodes.

- **Map Matching** This component receives a stream of Floating Car Data (FCD) and performs a road speed inference operation on disjoint subsets of the stream, belonging to short and anonymous trajectories of individual cars.

  While the main map matching kernel is computationally intensive and not easy to distribute to multiple nodes (instead, it should benefit from FPGA acceleration), the FCD stream can be split into disjoint parts and sent to multiple nodes, which will then execute the kernel independently of other nodes.

- **Neural Network Inference** The inference component of the traffic use-case should be able to perform thousands of inferences per second in a high-throughput mode. A challenging factor here is that there is no single machine learning model. Instead, separate models are needed for different roads within a city, and thus at any given time there might be tens of thousands of models that have to be ready to perform inference. There are nontrivial costs associated with (re-)loading the models' weights into CPU (or FPGA) memory if they cannot all fully fit there.

  The main metric to improve is the throughput in terms of inferences per second. We expect FPGAs to greatly contribute to speeding up the inference process. To further improve throughput, batches can be created and distributed among nodes at the cost of a higher latency. The language and the tooling should allow exploring this trade-off.

4.2 Requirements

HyperQueue, an HPC-oriented task runtime system, will be used to describe and execute coarse-grained workflows in a batch fashion. While it can already be used to execute task graphs and workflows on a distributed cluster, it has to be extended to accommodate the needs of the traffic simulator use-case. It will require the specification of fine-grained resource requirements for each executed workflow, especially focused on selecting which tasks require the use of an FPGA. This information should be then taken into account by HyperQueue in

order to optimally schedule the workflow on a heterogeneous cluster. Since some of the individual workflows might be using intra-job parallelization across multiple nodes, HyperQueue should also improve its support for scheduling and working with multi-node jobs.

To distribute fine-grained computation in the individual components of the traffic use-case, an appropriate framework, named *EvKit*[1] will be developed within EVEREST. The distribution should be done by splitting computing to multiple stateless worker nodes that will not share any data using a gather/scatter pattern. To improve load balancing, a message queue might be used to allow balancing the individual computational requests across multiple workers and also to allow adding and removing computational workers dynamically, without affecting the executing workflow.

The computation of alternative routes and/or the Monte Carlo simulation should be accelerated by FPGAs. EvKit should include support for offloaded computational kernels to FPGAs attached to host CPUs. It should also be extended with support for auto-tuning (using the mARGOt [4] auto tuner), to efficiently select which kernels should be executed on CPU or FPGAs, and also to decide which version of an FPGA kernel should be used to achieve maximal efficiency. These extensions will benefit the traffic simulator, the map matching and potentially also the neural network inference components of the traffic use-case.

To facilitate the integration of FPGAs kernels into individual components, the Ohua [3, 2] framework will serve as an abstraction layer for expressing heterogeneous programs. This shall additionally allow leveraging existing dataflow graph optimizations in the frameworks' compiler. Within EVEREST, the compiler framework will be extended to generate interfacing code for communicating with offloaded kernels running on an accelerator. Furthermore, dataflow graph transformations need to consider hardware restrictions for offloaded kernels. SDK integration will require adapting input and output formats used by the compiler from high-level languages like Rust to intermediate languages in Multi-Level Intermediate Representation (MLIR).

[1] More about *EvKit* in Deliverable 5.2 Sec. 2.1.

5 Kernel Computations

As outlined in Section 3, air quality modeling and renewable-energy prediction heavily rely on the WRF weather model. In general, most of the computational power required by these applications is spent solving differential equations using numerical methods. Due to the strong coupling of these methods in WRF, we added the Helmholtz decomposition use case to EVEREST to be able to show this for a fluid dynamic application on a manageable scope. Apart from CFD, WRF also models microphysical and -chemical processes that particularly computationally intensive, such as radiative transfer. For the traffic simulation use case, particle-based simulations were thought to be an option at proposal writing time. As already mentioned in Deliverable D2.2, particle-based abstractions were not deemed necessary after an analysis of the use cases and are not treated in this document. In addition to physics simulation, all the three uses cases of the EVEREST project use simulation data to drive machine learning algorithms. Such algorithms are dominated by linear algebra operations that have often been shown to lend themselves to acceleration.

Kernels in physic simulations or machine learning can be conceptually connected within a dataflow graph. This allows applying optimization techniques on the kernels themselves without needing to consider the whole application. The kernels we find in these applications can be interpreted as subsets of general tensor algebra. This view is often closer to their actual physical or mathematical formulation, also allowing them to be significantly more terse. These descriptions can also carry crucial expert knowledge about the problems they encode, which is lost in lower-level programming languages. This enables more impactful abstract transformations as oppose to structure-oblivious standard optimizations. Our use cases shall profit from specialized language and compiler support for these numerical stencils and general linear algebra kernels.

The rest of this section further explains optimization potential in the kernels that underlie the WRF model and the traffic simulation machine learning algorithms, as they provide our most common form of expert knowledge. For these kernels, we apply tensor optimizations that are widely discussed in the literature, in addition to EVEREST platform-specific transforms.

5.1 Problem Description

The Weather Research and Forecasting Model (WRF) [11] is a program for producing climate predictions and weather forecasts. It is an integral component in at least two of the three use cases of this project and is also used worldwide, making the contributions within EVEREST impactful beyond the project itself. The model code has a modular structure, allowing different components to be enabled and used as necessary. Each module concerns itself with a different aspect of a weather simulation, like cloud generation and movement, microphysics phenomena, or radiation calculations. These kernels are then called in regular intervals during the simulation, allowing them to update their respective model parameters. Depending on the complexity of the numerical computations done within the kernel, the execution times for the modules vary.

As a result, modules that take especially long to execute, most prominently the cloud movement, radiation and microphysics modules, are updated less often to improve latency. This has the side-effect of yielding less accurate data, as certain variables in the model are only accurate with respect to long-term trends. In practice, this leads to the simulation failing to capture weather phenomena associated to highly spatio-temporal processes, such as convection resulting in thunderstorms. Accelerating these complex modules is a key goal within EVEREST, partly because it opens up new possibilities for the project's use cases.

Augmentation or outright replacement of simulation elements in Weather Research and Forecasting (WRF) is only feasible at the driver level, which are decoupled subprograms with fixed interfaces. In EVEREST, we have decided to focus on the radiation driver Rapid Radiative Transfer Model for GCM Solvers (RRTMG), as it constitutes a significant portion of the runtime and coupling complexity, but is otherwise an ordinary tensor program. In this context, we are also able to compare against other state-of-the-art solutions, and bring their algorithmic improvements back to WRF.

Traffic modeling starts with the big data collection of FCD from mobile devices. These raw data records are map-matched to compute reference speeds on road segments for multiple time instances along a day. The processing of one day data (e.g. 20 million records for Prague) takes up to 4 hours on CPU 2-core

architectures. We target the execution duration under 1 hour while being energy efficient too. The GPS-projection, Dijkstra and Viterbi forming Hidden-Markov chain calculation has been identified as the bottleneck and thus the candidate for the kernel to be optimized.

The road speeds enter traffic simulator, which runs simulation mimicking the whole day dynamics aligned with the measurement data. In a nutshell it refines road speed data as well as it boosts information on weakly covered road segments the result of which is macroscopic traffic view on a city as well as training sequences for prediction calculation for major roads (e.g., 10,000 road models in Prague). The day dynamics simulation (on average 50,000 vehicles) takes approximately 20 hours, so the acceleration is needed as our target is to have this calculation done overnight (up to 4 hours).

Traffic management service relies on a large number of predictions of traffic for individual road segments. The predictions for each segments are computed individually using neural network models. To predict the traffic for a given time period for complete cities, inferences of on average 10.000 (depending on a size of a city) road segment models are required, which takes more than 1 minute on 2-core CPU architectures. Therefore, it is important to accelerate this inference calculation to provide traffic prediction for smart routing within seconds to become usable.

5.2 Requirements

In order to improve the execution time of computational kernels, the numerical computations they encompass could be described at a higher abstraction level. Similar work has been done for Computational Fluid Dynamics calculations in the past by members of the project [10, 12]. By using a high-level DSL, the compiler has more semantic information about the kernel, enabling data layout transformations and loop schedules that are no longer evident coming from lower-level languages like C or Fortran. In the context of this project, existing DSL abstractions are being extended to support the RRTMG algorithm. Furthermore, the compiler is targeted to generate code for the EVEREST platform, which includes both for multi-core CPUs and FPGAs.

We divide the requirements for the language into two. First, we discuss how the domain-specific abstractions can be embedded into the use case code (embedding, cf. Figure 3). We then discuss requirements on the abstraction itself (mapping, cf. Figure 3). Finally, we briefly discuss requirements to express ML kernels, an issue that is already well supported in ML frameworks.

5.2.1 Contextual Requirements

Standalone DSLs can provide great performance improvements. In reality, they must seamlessly integrate with the development environment and respect constraints imposed by the surrounding code. This includes code surrounding the kernel itself (e.g., in Fortran or Python) and other components of the programming stack, such as the language runtime, the operating system and the virtualization layer (from WP5). More concretely:

- The language provides a canonical interface that is interoperable with the Fortran ISO C binding. The DSL and compiler must represent and generate code for a subset of Fortran memory layout compatible data structures and calling conventions.
- The user must be able to introduce customization points in the original code base. The toolchain should assist the user in doing so through the use of a C preprocessor based mechanism. It must also provide a facade that can be integrated into a typical Unix build system, such as Make, that unobtrusively augments the existing build system.
- In heterogeneous implementations, memory transfers are often needed, most commonly at the point where execution forks to a different device. The language should use abstractions that allow the compiler to automatically insert such transfers. This should to a large extent be transparent to the programmer.

It is of high importance that these requirements leave the toolchain with as much freedom to implement a kernel as possible, opening up more opportunities for optimization. This includes mapping decisions of

computations to cores, threads and FPGA overlays as well as data to memories. Consequently, a contextual requirement for the DSL is to be platform agnostic. It should avoid asserting particular target devices or programming language that it is embedded in. Retargeting within the compiler should account for platform specific optimizations.

5.2.2 Application Requirements

Defining and implementing a DSL is always a choice made based on the observation that there exists a divide between the way requirements and goals are laid out in the application domain as opposed to the actual implementation in some more general programming language. What should and should not be part of a DSL depends on how much of the whole application is deemed relevant. Given the use case, this yields the following application-specific requirements:

- The DSL must have first-class support for expressing the mathematical expressions that make up our target application domain. For numerical simulations of differential equations, the language should support linear and tensor algebra. Support for stencil operations is also required, i.e. it must provide a full linear algebra abstraction. These can be divided into the following categories:
  - Element-wise arithmetic (`add, sub, mul, div, mod`)
  - Index reassociations (`diag, transp, proj`)
  - Regular constructors (`stack, expa, pad, rep`)
  - Fundamental operators (`prod, red`)
  - Complex operators (`contr, conv`)
- The DSL will be implemented on top of the MLIR framework, as a plug-in dialect. Domain-specific optimizations will be provided as part of a tensor program dialect created within this project, placed in the DSL lowering chain. Hardware-specific transforms and concepts, such as offloading and runtime management, will also be provided as MLIR dialects, independent of the DSL. This separation shall enable the reuse of existing MLIR abstrations to support additional workloads and optimizations. Lowerings to LLVM-IR will be provided, which allows targeting CPUs and FPGAs via the Bambu synthesis tool.

In essence, the chosen mapping of domain-to-language elements should provide a considerable benefit to the user over an implementation in a non-specialized language. This can be achieved through adopting more concise notations, reducing the amount of boilerplate code and ambiguity. To improve performance and other execution metrics, the compiler must be made aware of application specific and expert knowledge through the language.

5.2.3 Requirements for ML Inference

One key aspect for EVEREST is the interoperability with existing standards. Therefore we use existing DSLs where applicable. High-level operator abstractions are an established practice in the ML domain, where the Open Neural Network eXchange (ONNX) community standard is widely used. Depending on the details of the use case, it may be required to extend ONNX with support for quantized data types.

5.3 Challenges

One of the key challenges in transforming kernels of numerical simulations will be the stability of the result. In the past, there were attempts to employ Advanced Vector Extensions 2 in WRF computations to improve the execution time. This, however, led to highly unstable results and possibly also in numerical code crashes at seemingly random times, with errors not reproducible at the same times, which were attributed to the accumulation of floating point inaccuracy. Hence, special care must be taken when optimizing floating point operations in kernels, e.g. by comparing the results produced against unoptimized execution results or another form of

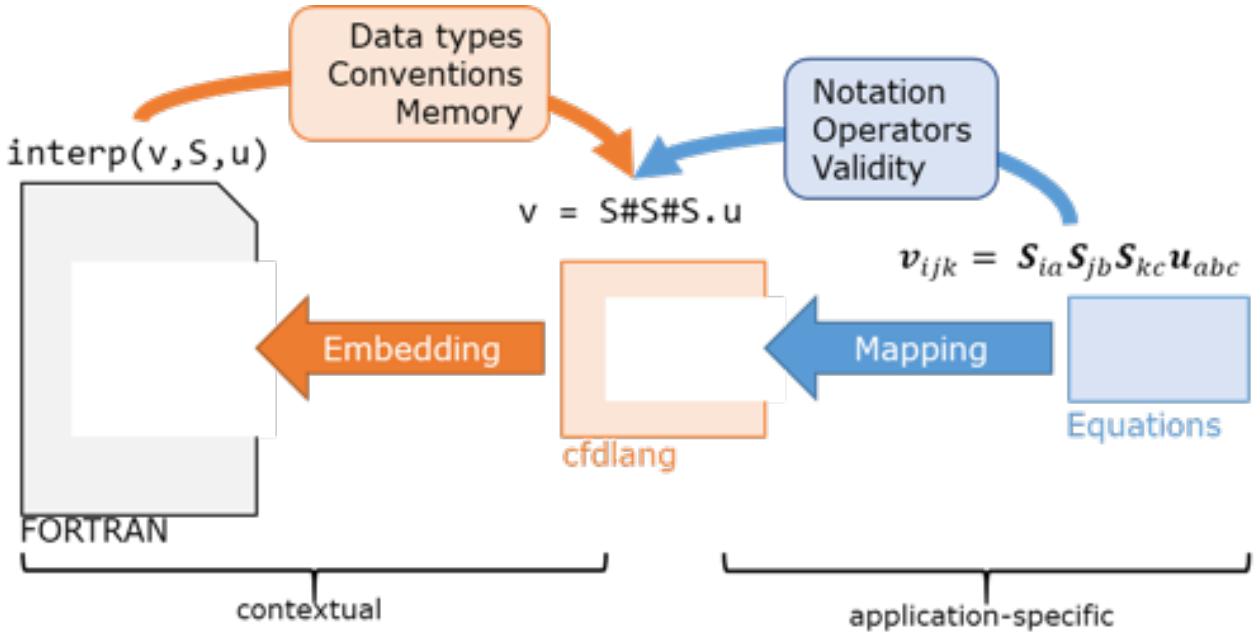

Figure 3 – Requirement factors for the Kernel DSL

gold standard. Other forms of validation that warrant a physical interpretation could also be aided by a DSL compiler, though most responsibility is left with the user. Methods for symbolic analysis of error propagation should be also considered.

Another challenge will be the integration of the DSL into the existing Fortran compilation flow to ensure that the code produced will link seamlessly to the rest of the WRF model. This includes integration with the runtime system, mapping data and synchronizing data transfers. The latter questions will be dealt with in a future report, once the hardware platform, the virtualization environment and the interfaces are more precisely described.

In addition to being reconfigurable, some drivers in the WRF model greatly vary with the evolution of runtime variables. The EVEREST framework should include provisions to adapt to the application workload. The compiler, in particular, should at least be able to produce different variants of the code to enable runtime selection. Should this be insufficient, code generation at runtime in a “Just in Time” manner will be considered. In this case, the overhead of just-in-time compilation and synthesis should be constrained so as not to considerably impact the overall application execution time.

For the ML kernels, there exist two major challenges: The first is the expression for custom data types, especially different floating and fixed point formats. This is important to improve the efficiency and latency of the accelerated kernels. At the same time, the accuracy of these kernels is very sensitive to details of the used data type. Here, it is challenging to express the necessary details at a high level, so that the application developer can interact with this description, while support the derivation of the optimal low level details during compilation of the accelerated kernels. This aspect becomes more difficult due to the fractured landscape of data type formats and representations currently used in the community. Here, EVEREST will find a unified expression that suits the use case but also fits to the wider ecosystem and the different involved compilation and synthesis tools. More information on data formats can be seen in Deliverable D2.6.

The second challenge is the expression of scaling and dynamic scheduling of the ML kernels. For the traffic prediction use case, a complete model could contain more than 5 billion parameters. But for typical inference requests only a subset of one model is needed to calculate the prediction. Here, a efficient yet practical way to express this dynamic dependencies need to be found.

6 Hardware Design Considerations

6.1 HLS Problem Description

FPGAs are increasingly becoming an attractive alternative target, providing valuable efficiency tradeoffs. One key opportunity afforded by reconfigurable devices is the possibility to continuously adapt the accelerator architectures to new algorithms, models, and iteratively provide optimizations without the need to change the device, coping with the exponential growth of algorithmic research in the area. However, a significant limitation in using FPGA devices is the requirement to develop the architectures in low-level hardware design languages (such as Verilog or VHDL), which is complicated and time-consuming. Traditional software languages allow the description of sequential instructions that do not depend on the low-level hardware implementation, while languages such as VHDL or Verilog require a good knowledge about digital design and circuits to produce efficient results. Expecting use case developers to follow the design of an application from the algorithm definition down to the FPGA programming is not realistic.

For these reasons, a more suitable approach for the acceleration of complex applications on FPGAs is to exploit HLS. HLS is a process that automatically translates high-level descriptions into hardware description language. The use of HLS tools raises the level of abstraction and automates the most time-consuming step in the development flow. Instead of manually writing VHDL/Verilog code, the user only needs to provide a program written in a standard programming language such as C/C++. The Register Transfer Level description generated by HLS represents an accelerator with standard interfaces that can be integrated in more complex system-on-chip architectures.

6.2 HLS Challenges

Current HLS tools have been developed mainly to generate efficient accelerators for regular, easily partitionable, arithmetic-intensive workloads typical of digital signal processing. They target extraction of instruction-level parallelism (sometimes also of parallel tasks, e.g., through OpenCL annotations) and consider simple memory subsystems. The synthesis of big data workloads introduces different requirements and challenges to HLS that will be addressed by EVEREST.

HLS tools typically assume that the generated accelerators will operate on local data that is available in on-chip memories, considering known and fixed memory access latencies and performing optimizations that aim to reduce such latencies. As a consequence, they do not adequately serve applications that operate on large datasets that cannot fit into on-chip memories or even external accelerator memory. Moreover, they only consider fine-grained memory parallelism, and they do not handle well data-dependent operations, highly unbalanced parallel activities, or synchronization through atomic memory operations.

HLS-based solutions for optimizing the memory accesses and exploiting coarse-grained parallelism will make the EVEREST approach amenable to the power efficient execution of workloads with large datasets. In addition, accelerators will become more efficient with proper optimization of memory accesses and data transfers through custom memory architectures outside the computational logic, using intelligent memory managers automatically generated with a combination of compiler information and hardware generators. In EVEREST, this will require both tools for the generation of complete memory architectures to complement existing HLS methodologies. The interfaces between the accelerators and the rest of the platform (see Section 6.3) will be based on the Advanced eXtensible Interface (AXI), part of the ARM Advanced Microcontroller Bus Architecture (AMBA) specifications. Such interfaces allow us to integrate the accelerators generated in EVEREST within the platforms on the target nodes.

To further improve energy and latency savings, EVEREST will also extend the support for floating-point computation with variable precision within the HLS flow. For example, in Machine Learning/AI-based tasks, variable precision operations can reflect different quantization techniques used in deep learning algorithms.

6.3 FPGA-based Target Platform

In the EVEREST project two different major workload types are observed in the use cases, i.e., single-location heavy computational workloads and distributed workloads in loosely coupled systems. Throughout the project we are going to selectively decide which processing parts of those workflows can benefit the most from the specialized language as well as the heterogeneous EVEREST platforms.

Those platforms may feature one or more FPGA devices for hardware acceleration and one or more physical memories (either local or external to the FPGA), as shown in Figure 4. Such systems will run Linux as Operating System (OS) and a hypervisor to manage the hardware resources. Note that the EVEREST approach is not limited to these architectures. In fact, specifying the workflow pipelines at a higher level of abstraction allows us to easily port the applications to architectures with heterogeneous Graphics Processing Unit (GPU)-based nodes and end-user embedded devices.

To examine the potential of programmable heterogeneity in EVEREST workflows, we propose the employment and extension of two state-of-art research platforms that leverage FPGAs in different architecture configurations. The first is a CPU-managed system that rely on tightly-coupled, bus-attached FPGAs. The second is an FPGA-disaggregated system that relies on loosely-coupled, network-attached FPGAs.

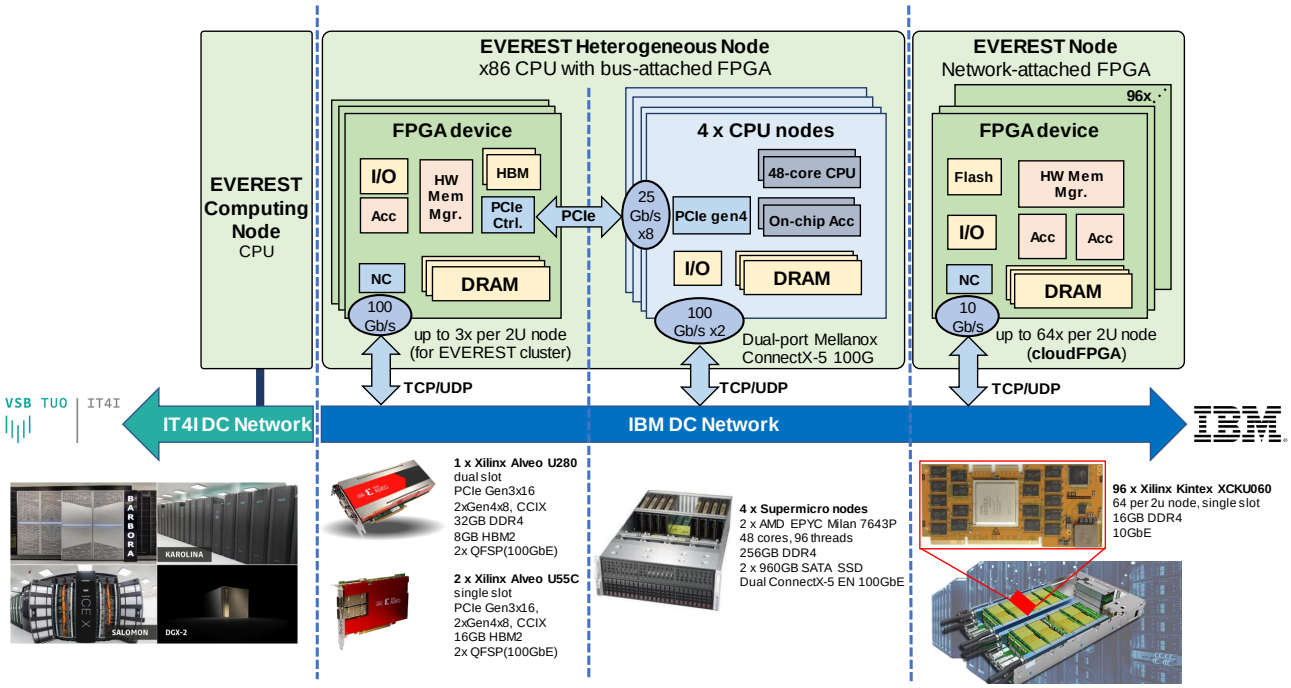

Figure 4 – EVEREST experimental heterogeneous platforms

Both those systems abstract the way that the FPGA accelerators are being developed and integrated and offer high flexibility to the EVEREST consortium to account for interoperability and retargetability of the developed accelerated solutions to different platforms (even out of EVEREST's platforms).

This abstraction is enabled by a predefined set of interface requirements. There are mainly two considerations tied to those requirements, a) the interface of the accelerators to the host through a software API and b) the interface of the accelerators inside the FPGA at the RT-level. Both interfaces are offered by the Integrated Development Environments (IDEs) of the two platforms, i.e., the Alveo platform for the bus-attached FPGAs and the cFDK of the cloudFPGA research platform.

As shown in Figure 5, the accelerators in both platforms are interfaced through AXI channels. Both Alveo platform and cFDK provide a full AXI master bus to communicate with the host via the FPGA Shell. cFDK also enables AXI-stream based access. Same AXI master buses are used to connect the accelerators to the FPGA DRAM channels (also High Bandwidth Memory (HBM) for Avleo-based FPGAs).

The Shell logic of both platforms implement all the necessary low-level processing of the Peripheral Component Interconnect express (PCIe) and TCP/IP respectively, in order to provide those AXI interfaces. This

way the developers can generate the accelerators with only this interface requirement. Such interfaces are standardized and commonly used in the FPGA design ecosystem, while they can be generated by HLS tools with `#pragma` directives at the function definition level.

On the host software side, the two platforms offer different Application Programming Interfaces (APIs). The Alveo-platfrom relies on the Xilinx Runtime Libraries and its corresponding PCIe functions (XOCL and XCLMGMT). On top of them a C/C++ interface is provided and based on that, different language porting can be done, e.g., Python through a C-to-Python tool (e.g., SWIG, Pybind11 etc.). On the other side, cFDK offers the seamless connection to any TCP/UDP socket and thus any programming language or library compatible with sockets can be interfaced directly.

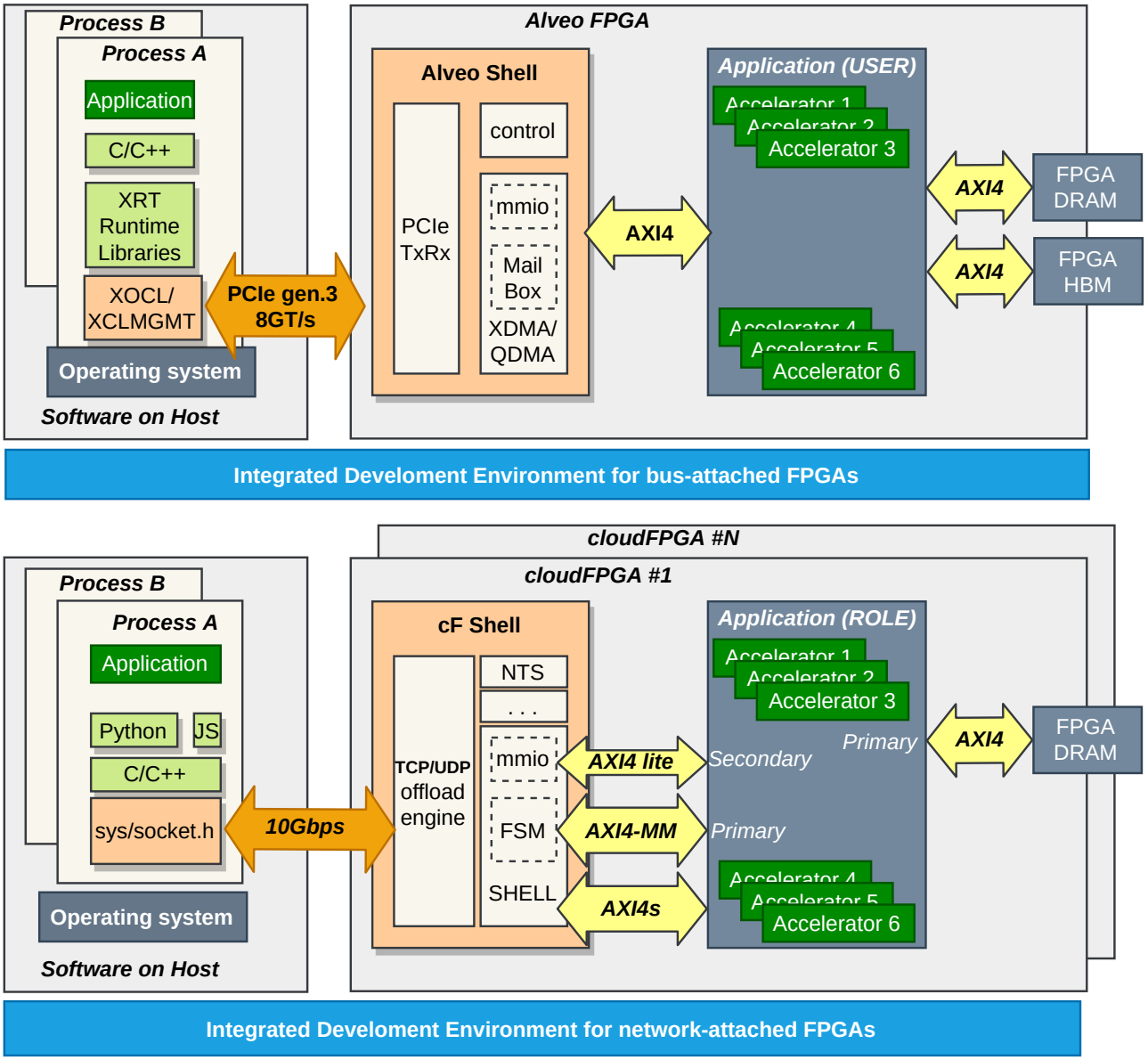

Figure 5 – Interface requirements for FPGA accelerators a) at the host software side and b) at the FPGA side for both Alveo (top) and cFDK (bottom)

7 Use Case and Framework Requirements

This section summarizes important observations extracted from the use cases that are important from the point of the view of the programming framework. From these observations we derive general requirements for the components of the use cases in terms of programmability and interoperability, among other properties. To account for these properties, we distill requirements for the components of the programming framework and their interfaces, including language abstractions and annotations, compiler support, runtime support, and platform support.

7.1 Summary: Properties of the Use Cases for Programming Support

Table 1 describes high-level properties of the use cases as a whole. As can be seen, the use cases represent a challenging combination of HPDA, HPC and ML components, stressing today and future programming frameworks. At the higher-level, use cases are distributed across different geographical locations, while requiring efficient coordination for distributed computing within a site (e.g., via HyperQueue). The use cases are implemented in multiple languages, making language integration an important requirement. As discussed above, all use cases have components that require batched processing, with traffic modelling requiring both streaming and batched processing. Similarly, all use cases are time critical, in the sense that results delivered too late are either irrelevant (e.g., a prediction of something in the past) or can potentially lead to economic costs (e.g., in the case of prediction for renewable energies). All three use cases use ML techniques for decision making, with inference possibly offloaded to the edge.

	Distributed	Multi-location	Multi-language	Streaming	Batch	Time criticality	ML components	Edge-enabled
Renewable-energy prediction	X	X	X		X	X	X	
Air-quality monitoring	X	X	X		X	X	X	X
Traffic modeling	X	X	X	X	X	X	X	X

Table 1 – High-level properties of EVEREST use cases

At a finer granularity, use cases have to profit from the novel computing nodes proposed in EVEREST. This requires a detailed analysis of individual components within the larger workflows. Properties of the main components are shown in Table 2. These components are selected for being critical for the execution of the use cases. Unless the envisioned target platform is FPGA, not all of them will take advantage of the Everest technologies. Less critical components Components not included in the table will receive standard support by the high-level platform, meaning no special language or framework support to improve execution metrics or programmability. Table 2 depicts a heterogeneous landscape for tool support, which imposes requirements on use case providers, language and framework design, and tool interfaces, as will be discussed in the following. Some of the components are being developed at the time of writing. In this case, we report on what is planned whenever possible.

7.2 Requirements

EVEREST as a whole contributes to different aspects of system design and programming. Global requirements and a detailed representation of the degree to which the requirements should be met for the different components are presented in Table 3. The table includes the following requirements:

ID	Name	Lan-guages	Target resource	Compute class	Data class
C1	WRF Assimilation	Fortran	CPU	HPC, I/O bound	Regular data
C2	WRF data preparation	Fortran	CPU	HPC, I/O & Memory bound	Regular data
C3	WRF radiation	Fortran	CPU, FPGA	HPC, compute-bound	Regular data
C4	WRF cloud movement and microphysics	Fortran	CPU, FPGA	HPC, compute-bound	Regular data
C5	Energy Production modeling	Python	CPU	Cloud, Edge	Irregular data
C6	AirQuality modeling	Python	CPU	Cloud, Edge	Irregular data
C7	Traffic: Map matching	C++	CPU, FPGA	Cloud, I/O and storage bound	Regular data
C8	Traffic Prediction: AI Training	Python	CPU, GPU	HPC, Compute intensive	Regular data
C9	Traffic Prediction: AI inference	Python	CPU, FPGA	Cloud, Edge	Regular data
C10	Traffic: Intelligent routing (PTDR)	C++, Rust	CPU, FPGA	HPC, Cloud	Irregular data
C11	Traffic simulation	Python, Rust, C++	CPU, FPGA	HPC, Compute intensive	Irregular data

Table 2 – Properties of key components in EVEREST use cases

ID	Global requirement	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
GREQ1	Programmability											
GREQ2	Interoperability											
GREQ3	Retargetability											
GREQ4	Performance											
GREQ5	Energy efficiency											

Table 3 – Global requirements on key use case components (the darker, the more important the requirement is)

GREQ1 Programmability: End users of the platform should transparently profit from the EVEREST platform. This means that with minor effort, programmers can have functionality executing on, for instance, an FPGA without having to write a single line of code in an Hardware Description Language (HDL). Code modifications include annotations or inserting DSL expressions in existing code. As an example, as discussed in Section 5, the main numerical components of the WRF model (C3 and C4) will profit from expression DSLs to automatically create FPGA accelerators for stencils and other linear algebra operations. Machine learning components (e.g., C5, C6) do not require that much programming support if the target architecture is not FPGA, since this is already accounted for in machine learning frameworks.

GREQ2 Interoperability: End programmers use different frameworks (cf. Table 2) and languages. For instance, a learned model must be exported from the framework to be deployed on the platform. This requires support for standard formats and for clearly defined interfaces within the EVEREST programming framework. Enforcing interoperability is a key challenge for EVEREST, also for collaborating with other ICT-51 projects (see the events driven by BDVA). This will significantly increase the long-term impact of the EVEREST toolchain.

GREQ3 Retargetability: The EVEREST computing platform scales from the edge all the way to the data center. Different instantiations of the platform, as well as alternative technological options (e.g., Xilinx FPGAs and HLS tool flows), have to be supported by the EVEREST programming framework.

GREQ4 Performance: The EVEREST programming frameworks shall help improve the performance of applications. Naturally, the development will focus on the more performance-critical components of the use cases from Table 2.

GREQ5 Energy efficiency: Apart from performance, an increasingly important property of systems is the energy efficiency. By clever use of programmable and reconfigurable resources, the EVEREST programming environment must have energy efficiency as the second objective.

From these global requirements and the information in Table 1 and Table 2, we distill requirements for different flows to be implemented in the EVEREST software development kit. The flows and the requirements are described hereafter. In addition to that, Table 3 has been used also to select the main components of the use cases to be used for the evaluation phase.

7.2.1 Overall Envisioned Flow

Figure 6 provides an overview of the envisioned EVEREST programming environment. We first discuss the role of the different components before listing their requirements in Section 7.2.2–Section 7.2.6

7.2.1.1 Workflow Distribution and Parallelization

The preliminary use case analysis in Section 3 showed that some use cases include highly computing intensive workflows. The traffic simulation use case has been identified as particularly amenable for optimization through distribution. Within the project, HyperTools [1] shall be used to orchestrate these large application flows. HyperTools is a platform used to define and execute workflow pipelines in large-scale distributed environments. Using a simple Python interface, end-users can define their application flows, which are then executed on a HyperTools server that distributes the work onto several workers.

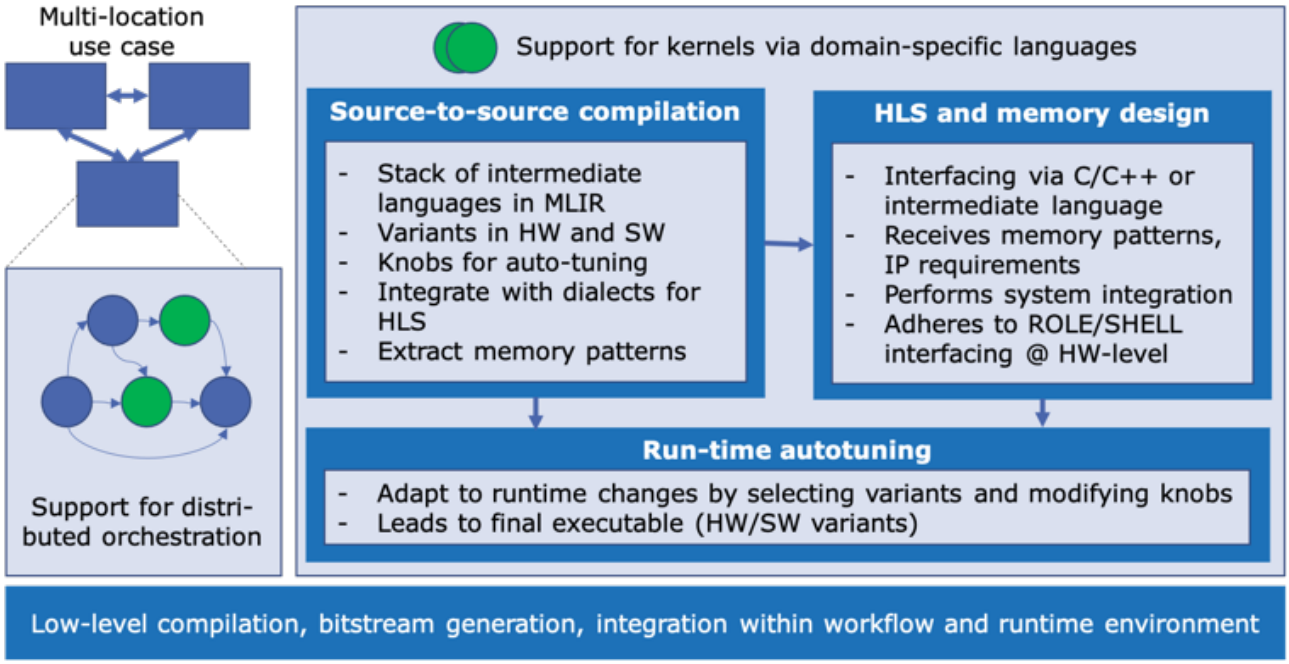

Figure 6 – Envisioned overall flow

As discussed in Section 4.2, a batch-enabled pipeline framework in EVEREST needs to efficiently distribute shared data between consequent simulated iterations or reuse it across workflows.

To combine kernels executed on accelerators with conventional CPU code more easily, the Ohua compiler framework will be used, as outlined in Section 4.2. Its DSLs will be used to describe the high-level algorithms that forms an application. This algorithm may include functions that are offloaded as kernels to accelerators. The compiler will transform the algorithm into a Dataflow Graph (DFG), potentially apply various graph rewrites and produce a runtime that executes the graph, linking host machine and accelerator.

7.2.1.2 Embedded DSLs

Embedded DSLs offer great potential to simplify coding while opening up more possibilities for optimization. Given the preliminary analysis discussed in Section 5, we will extend prior DSLs for CFD and tensor-based computation, e.g., CFDLang [10], TeML [12], and TeIL [9]. The latter works by constructing an AST in-place in the code. This level of control is particularly important to control memory access patterns. We aim for a clear-cut call-like embedding that leaves us with the freedom to rearrange and possibly precompile. Additionally, in order to disseminate our DSL and obtain feedback, we have considered building library targets that could also be integrated with existing projects right away.

7.2.1.3 Multi-level Intermediate Representation with MLIR

One of the explicitly stated goals of the project is to achieve an interoperable tool flow. To that extent, we will build on the steadily maturing MLIR framework [5]. MLIR is working towards compatibility with a wide variety of existing back-ends, hinting at possible vendor support in the future. In addition to this, MLIR is also being used for hardware specification within the EVEREST consortium and outside (e.g., CIRCT[2]). With MLIR, we envision a modular compilation pipeline, leveraging the design effort of the open-source community. By designing EVEREST dialects that the DSL abstractions map to, we can profit from the existing lower-level dialects for linear algebra.

[2] https://github.com/llvm/circt

7.2.1.4 HLS and Memory Design

HLS allows application designers to accelerate specific kernels on FPGA without having hardware design experience. Moreover, since HLS uses high-level input languages (e.g., C/C++), the system-level integration is simplified. In addition to the acceleration of the kernels, as stated in Section 6.2, an HLS flow allows for the optimization of the energy consumed by the EVEREST applications. EVEREST will consider two alternative HLS tools: Xilinx Vitis HLS and Bambu [7]. Xilinx Vitis HLS is one of most common HLS tools, composed of an open-source frontend based on the LLVM compiler and a closed-source backend. It supports a wide range of optimization directives and standard accelerator interfaces. Bambu is an open-source HLS tool developed by project partners, supporting inputs written as C/C++ specifications, also annotated with OpenMP parallelization directives, and compiler Intermediate Representations (IRs) coming from the well-known Clang/LLVM and GCC compilers. The broad spectrum and flexibility of input formats allow the seamless integration of several source-to-source compilers (like MLIR) and design space exploration frameworks. Bambu already includes many hardware-oriented optimizations and interfaces with logic synthesis and verification backends, either commercial or open-source, allowing to develop accelerators for targets different from Xilinx devices. It provides a modular toolflow with multiple opportunities to explore new synthesis methodologies, optimizations, and architectural features dedicated to big data applications.

Since the EVEREST applications have a strong focus on data management, EVEREST includes a specific flow to customize the memory infrastructure around the accelerators. For doing this, we aim at extending Mnemosyne [8], an open-source CAD prototype for the customization of memory architectures. Mnemosyne currently supports the creation of multi-port, multi-bank private local memories that can interfaced with HLS-generated accelerators. It will be extended with the support for more memory-related components for the creation of “intelligent memory managers” that are optimized based on the information extracted during the compilation flow. We are also developing Olympus, a prototype flow for automatic system-level integration of accelerators and hardware components that directly interfaces with the high-level flow.

To accommodate for the unique DSL and characteristics of ML inference workloads, EVEREST includes a customized flow. This new flow imports a community standard (ONNX) analyze the compute graph of the ML model with focus on the specific performance metrics of the target platform. Afterwards, the model is partitioned across multiple FPGA nodes, if required and the most efficient architecture is generated to meet a users performance and resource constraints. After this Domain-Space Exploration (DSE) the generated (partial) architectures for the compute kernels are handed over to other EVEREST tools, e.g. Bambu for kernel synthesis and Olympus for memory management.

7.2.1.5 Run-time Auto-Tuning

Computation times in several use cases can fluctuate, based on the input data, while optimal kernel configuration or their versions can heavily depend on the target architecture and the available resources. To adapt to these changes, the toolchain includes the mARGOt dynamic autotuning framework [4]. This will help to ensure that the application meets its timing constraints.

mARGOt allows the programmer of an application to expose software-knobs, which can be used to influence the execution of the program. Using a pre-defined set of objectives (e.g., “Achieve a specific throughput with as high accuracy as possible”), the auto-tuner will use the exposed software knobs to meet the objectives as best as it can. In the context of kernel computations that include multiple versions, mARGOt could be used to regulate the execution frequency of computationally intensive kernels or to trade accuracy in the results for higher throughput, e.g., by reducing the polynomial degree of interpolation operations or the number of samples in Montecarlo simulations.

7.2.2 Requirements: Orchestration Large Application Flows (DAGs)

The *REQ2.2* has been revoked. It has showed up that intense sharing of a global state between computational nodes has a significant negative impact on the performance; it holds even for exchanging updates only. The

distribution is going to be solved as a part of each use case as it is problem-related rather than a general pattern.

Table 4 – Requirements for EVEREST HyperTools extensions

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ2.1	Front-end for EVEREST Applications	Framework on top of HyperTools for easy use-cases driven development	Tool	Must have	Specific interface for distribution of tasks in traffic simulator	GREQ1
REQ2.3	API for communication with virtualization environment	The goal is to establish a way of communication between the scheduler and the environment to exchange all important properties and constraints.	API	Must have	If a dynamic reconfiguration of the environment is possible, the protocol have to be able to notify the scheduler about the changes	GREQ3

7.2.3 Requirements: Language and Compiler

Table 5 – Requirements for language and compiler support

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ3.1	WRF Expression abstraction	Language support for expressions in numerics (tensors, linear algebra)	Method-ology	Must have	Kernel support for WRF simulations.	GREQ1
REQ3.2	WRF Fortran integration	Expression abstractions should be callable from within Fortran code	Method-ology	Must have	Either by annotations or inline code modifications, user can write expressions within Fortran code for WRF	GREQ2
REQ3.3	ML integration	Framework integration with machine learning frameworks	Method-ology/ API	Should have	Allow importing models to hook into the code generation process for EVEREST specific transformations	GREQ2, GREQ4

Table 5 – Requirements for language and compiler support

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ3.4	Streaming support	Language support for streaming workflows with highly dynamic loads	Method-ology	Could have	Enable compiler reasoning for reconfiguring streaming oriented computations. Expected to support traffic use case. Will be revisited as the implementation-progresses.	GREQ1, GREQ4, GREQ5
REQ3.5	Integration with compiler frameworks	For stability, reusability and extensibility, compiler work should build on top of established frameworks (e.g., LLVM and MLIR for numerics, Haskell or alike for dataflow)	Method-ology/ tool	Should have	By contributing to open source frameworks, the results from EVEREST can be used by the community at large. By integrating with these frameworks, EVEREST can reuse and extend existing methods.	GREQ2, GREQ3
REQ3.6	Compiler transformations for kernels	At the middle-end, the compiler must include a framework for transformations to manipulate code and optimize for the EVEREST platform	Method-ology	Must have	For numerics, this should include affine transformations (polyhedral) with support for stencils and other linear algebra primitives	GREQ4, GREQ5
REQ3.7	Compiler transformations for dataflows	For dataflow programs, the compiler should include semantic preserving rewrites for performance and energy optimizations, while retaining determinism	Method-ology/ tool	Could have	This should extend on previous work on optimization for dataflow programs (including mapping, graph rewrites and I/O batching)	GREQ4, GREQ5

Table 5 – Requirements for language and compiler support

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ3.8	Multi-target code generation	The source to source compiler should generate code for different targets	Method-ology/ tool	Must have	Code written in high-level expression abstractions should translate to pure software (C/C++ code), or software with offloading to accelerators (e.g., FPGA)	GREQ3
REQ3.9	Generation of tunable parameters	To enable autotuning, the compiler must produce descriptors of solutions to interface with mARGOt.	Method-ology/ tool	Must have (for adapt-able kernels)	From high-level abstractions, the compiler should extract knobs and parameters that are key to modifying performance and/or energy efficiency	GREQ4, GREQ5
REQ3.10	Interface to HLS	The compiler should enable a downstream HLS flow	Method-ology	Must have	The compiler must export code (or an intermediate representation thereof) to the HLS flow, including behavioral descriptions of the kernels and configuration information for memory modules. The representation must be synthesizable (e.g., no dynamic allocation of memory).	GREQ4, GREQ5

Table 5 – Requirements for language and compiler support

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ3.11	cFDK/OC-Accel software integration and language compatibility	The software should be compatible with the cFDK/OC-Accel API	Methodology/ API	Must have	The software part of the kernels that are being mapped to the FPGA should be built in a way that allows the seamless integration with the API specifications of cFDK (e.g. C/C++/Python sockets) and/or OC-Accel frameworks (e.g. C/C++/libocxl)	GREQ2, GREQ3
REQ3.12	Reasoning about heterogenenous applications	Compiler optimizations must consider offloaded kernels	Methodology/ tool	Must have	The source to source compiler must differentiate between offloaded and host functions to apply parallelizing transformations to host code only.	GREQ2, GREQ3
REQ3.13	Glue gode generation for heterogenenous applications	Generate interfacing code for offloaded kernels	Methodology/ tool	Should have	The source to source compiler must generate code that allows the interfacing between host and accelerator.	GREQ2, GREQ3
REQ3.14	Abstractions for offloaded kernels	Provide abstractions for marking a kernel as ‘to be offloaded’ in a high-level algorithm	Methodology/ tool	Should have	The high-level dataflow language provides abstractions for marking offloaded kernels, easing development of the application.	GREQ1, GREQ2, GREQ3

7.2.4 Requirements: High-level Synthesis and Memory Design

Table 6 – Requirements for high-level synthesis and memory design

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ4.1	C/C++ support	C/C++ support for HLS of descriptions coming from DSL compiler	Method-ol-ogy/tool	Must have	The HLS tool should support C/C++ code from the use case applications.	GREQ1, GREQ2
REQ4.2	Bambu LLVM IR support	Low level integration with DSL compiler	Method-ol-ogy/Tool	Must have	The HLS tool should support a version of LLVM consistent with the one used by the DSL compiler.	GREQ2, GREQ4
REQ4.3	Bambu MLIR dialect support	Direct synthesis from MLIR dialects	Method-ol-ogy/tool	Can have	It may improve the final performance raising the abstraction level. At least, it should support integration with the affine dialect.	GREQ2, GREQ4
REQ4.4	HLS Verilog output	HLS generates RTL Verilog code as output	Method-ol-ogy/tool	Must have	The Verilog code must be synthesizable with respect the EVEREST backend flow.	GREQ2
REQ4.5	HLS VHDL output	HLS generates RTL VHDL code as output	Method-ol-ogy/tool	Should have	The VHDL code must be synthesizable with respect the EVEREST backend flow.	GREQ2
REQ4.6	Top function specification	The code to be synthesized must be in a stand-alone function which needs to be specified	Method-ol-ogy/tool	Must have	The top function could be specified as annotation to the code, command line parameter or option files.	GREQ2
REQ4.7	Block level/Top component interfaces	The protocol to interface with the top module has to be specified	Method-ol-ogy/tool	Must have	The start, done, etc. protocol of the top component has to be defined and compatible with the EVEREST platform.	GREQ2

Table 6 – Requirements for high-level synthesis and memory design

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ4.8	Port-Level interfaces	IO interface protocols added to the individual function arguments	Method-ol-ogy/tool	Must have	The definition of the protocol should be defined through code annotations.	GREQ1, GREQ2
REQ4.9	Bambu Vivado HLS IO interface interoperability	Annotations specifying the IO protocols interface compatibility	Method-ol-ogy/tool	Can Have	Block/port level interfaces should use the same annotations used by Vivado HLS.	GREQ2
REQ4.10	Technology options specification	The HLS tool accepts inputs for optimization, clock constraint and resource constraints.	Method-ol-ogy/tool	Must have	These technology constraints will passed as input options.	GREQ2
REQ4.11	Bambu Data flow annotations	HLS Data flow support	Method-ol-ogy/tool	Should/hav	Dataflow style applications could be specified by code annotations.	GREQ1
REQ4.12	Bambu OpenMP support	OpenMP for pragma synthesis support	Method-ol-ogy/tool	Can have	The body of OpenMP parallel loop needs to be in a separate function.	GREQ1
REQ4.13	Bambu floating point precision	Floating point variables may use a custom floating precision data type.	Method-ol-ogy/tool	Can have	Allow optimizations of scientific and machine learning kernels.	GREQ1
REQ4.14	cFDK/OC-Accel top component interface	Interface definition of the top component being intergraded with cFDK/OC-Accel frameworks.	Method-ol-ogy/tool	Must have	The top-level component of the functionality that will be mapped to the FPGA must be compatible with cFDK ROLE interface (AXIlite AXIs, AXIm) and/or OC-Accel Action interface (AXIlite, AXIm)	GREQ2 GREQ3
REQ4.15	Memory interfaces	Standard interfaces for memory access	Method-ol-ogy/tool	Must have	The HLS-generated kernels and the memory modules should have a common interface format	GREQ2

Table 6 – Requirements for high-level synthesis and memory design

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ4.16	Software-level support	Software code to interface with the accelerators.	Method-ol-ogy/tool	Must have	The accelerators should be invoked with custom OS drivers	GREQ1
REQ4.17	Hard-ware/software data sharing	Data allocation must be compatible with hardware memory interfaces	Method-ol-ogy/tool	Must have	The software-level data allocation should be performed in a way that hardware can access the data	GREQ2

7.2.5 Requirements: Autotuning and Virtualized Environment

Table 7 – Requirements for the Virtualized Environment and in particular Dynamic Autotuning

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ5.1	Application knobs	The autotuning framework should have access to the application knobs	Method-ology/ tool	Must have	The access should be provided by means of the DSL or by the application itself. The dynamic autotuning framework is only a decision engine.	GREQ4, GREQ5
REQ5.2	Adaptive autotuning	The dynamic autotuning framework should be able to adapt depending on decisions taken on the virtualized environment	Method-ology/ tool	Must have	The application adaptation should be triggered by changes in the available resources	GREQ2, GREQ4, GREQ5
REQ5.3	Integration with runtime	The dynamic autotuning framework should be able to interact with the virtualized environment	Method-ology/ tool	Could have	Given the knowledge of the application, the dynamic adaptation framework can provide hints to the virtual manager to steer decisions	GREQ2

Table 7 – Requirements for the Virtualized Environment and in particular Dynamic Autotuning

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ5.4	Autotuning and optimization	The dynamic autotuning framework should manage the software knobs exposed by the application by autonomously selecting a near-optimal configuration	Method-ology/ tool	Must have	The adaptation should be triggered automatically and not by hand	GREQ4, GREQ5
REQ5.5	Variant selection	The dynamic autotuning framework should be able to manage code variants selection	Method-ology/ tool	Must have	Code variants should be managed as for the parameters of the application	GREQ4, GREQ5
REQ5.6	Design and deploy-time information	The dynamic autotuning framework should be able to take a decision based on knowledge collected at design time or at deploy time	Method-ology/ tool	Must have	The knowledge for taking the decision can be directly injected by some analysis of the compilation flow or extracted by an on-line profiling of the kernel	GREQ4, GREQ5
REQ5.7	Language support	The dynamic autotuner requires C++ applications	Method-ology/ tool	Must have	mARGOt is a C++ library to be linked with the application	GREQ2
REQ5.8	HW Knobs	The dynamic autotuning framework should have access to HW knobs	Method-ology/ tool	Could have	In case of a configurable accellerator that can be dynamically configured, the knobs have to be exposed to the SW layer.	GREQ4, GREQ5

Table 7 – Requirements for the Virtualized Environment and in particular Dynamic Autotuning

ID	Name	Description	Nature	Priority	Comments	Relation to global
REQ5.9	HW monitors	HW accelerators should expose a monitor interface to the run-time to trigger dynamic decisions	Method-ology/ tool	Could have	Monitoring information should be exposed to the SW and will be used by the run-time environment (dynamic autotuning and virtualize environment) to take dynamic decisions.	GREQ4, GREQ5
REQ5.10	Execution	The run-time environment requires a CPU where to execute	Method-ology/ tool	Must have	The runitme environment is composed by software modules that requires a core where to execute. Pure HW environments are not considered.	
REQ5.11	Virtual Environment	Virtualization environment has to enable execution of applications on different hardware in terms of processor and accelerators	Method-ology / tool	Must have	The EVEREST virtualization environment targets different hardware accelerators and CPU architectures	GREQ2, GREQ4, GREQ5

7.2.6 Requirements: Use Case Providers

As mentioned in Section 5, one of the most important factors for pursuing specialized language support is enabling the encoding of expert knowledge. Our investigation into WRF and experience with past projects already show a variety of ways how expert knowledge can be leveraged for high-level optimizations. This implies that a close cooperation with domain experts is key for a successful language design. While we can make solid assumptions for many language features based on the observations we made for the kernels, such as the focus on linear algebra, much falls outside the scope of language and compiler development. Mappings and identities pertaining to the physical or application-specific interpretation are fundamentally the responsibility of the use case providers. It is them who have the authority over these tacit requirements placed on the compiler development.

At the same time, the success of the language design relies on the ability to encode and exploit them, which means that our development processes are tied together. While we have started this during the creation of this document, both the language design and the use cases are a moving target. We believe the requirements specified here are a great starting point for continuous collaborative development, as all sides continue to probe their design space.

8 Conclusions

In this deliverable we have discussed elements in the use cases that can benefit from language support within the EVERET SDK. Together with the appropriate tooling it will help programmers to transparently leverage the efficiency of the EVEREST heterogeneous platform. After reviewing the use cases, we discussed considerations for the SDK design and the hardware design of the EVEREST platform. This deliverable closed with a detailed analysis of the requirements of the use cases, and derived specific requirements for the individual components of the EVEREST SDK, i.e., orchestration of workflows, domain-specific languages and compiler, high-level synthesis and memory design, and autotuning and virtualized environment.

The analysis presented here reflects a highly heterogeneous use case landscape, combining different computational patterns, input languages, and build systems. Compared to Deliverable D2.2, this deliverable contains a complete specification of the use cases and a more detailed analysis of critical components of the use cases that require acceleration via the EVEREST SDK. Machine learning algorithms, exploited by traffic management, energy, and air quality pilots, are well defined and can be readily supported by existing frameworks. Important in the SDK will be the support of interoperability (reading in and processing models exported from different machine learning frameworks), interoperability with other application phases, and support for distributed execution on FPGA-based systems. Apart from machine learning, we observed two different major workload types, namely HPC (e.g., weather simulations) and coordination of loosely coupled tasks (e.g., data acquisition and data assimilation tasks). A special focus will be set in heavy computational workloads, to provide language and compiler support, so as to transparently accelerate portions of HPC applications. This will be enabled by tailor-made domain-specific abstractions coupled with runtime components which enables us to achieve high interoperability and retargetability at a low cost to users of existing code bases. For task coordination, we consider language support for implicit definition of task or dataflow graphs. This will be driven by the traffic use case. The availability of a powerful HPC facility, including FPGA resources, is strictly necessary for the execution of the weather simulations in support of energy and air quality machine learning algorithms.

This document serves as guide for the work in work packages WP4 and WP5.

References

[1] Vojtěch Cima, Stanislav Böhm, Jan Martinovič, Jiří Dvorskỳ, Kateřina Janurová, Tom Vander Aa, Thomas J Ashby, and Vladimir Chupakhin. Hyperloom: A platform for defining and executing scientific pipelines in distributed environments. In Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms, pages 1–6, 2018.

[2] Sebastian Ertel, Justus Adam, and Jeronimo Castrillon. Supporting fine-grained dataflow parallelism in big data systems. In Proceedings of the 9th International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM), PMAM'18, pages 41–50, New York, NY, USA, February 2018. ACM.

[3] Sebastian Ertel, Christof Fetzer, and Pascal Felber. Ohua: Implicit dataflow programming for concurrent systems. In Proceedings of the Principles and Practices of Programming on The Java Platform, pages 51–64. 2015.

[4] Davide Gadioli, Emanuele Vitali, Gianluca Palermo, and Cristina Silvano. Margot: a dynamic autotuning framework for self-aware approximate computing. IEEE transactions on computers, 68(5):713–728, 2018.

[5] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: A compiler infrastructure for the end of moore's law. arXiv preprint arXiv:2002.11054, 2020.

[6] OpenCAPI Consortium. A statement from the opencapi consortium leadership, 2022. https://opencapi.org, visited on 2023-01-23.

[7] Christian Pilato and Fabrizio Ferrandi. Bambu: A modular framework for the high level synthesis of memory-intensive applications. In 2013 23rd International conference on field programmable logic and applications, pages 1–4. IEEE, 2013.

[8] Christian Pilato, Paolo Mantovani, Giuseppe Di Guglielmo, and Luca P. Carloni. System-level optimization of accelerator local memory for heterogeneous systems-on-chip. IEEE Transactions on CAD of Integrated Circuits and Systems, 36(3):435–448, 2017.

[9] Norman A. Rink and Jeronimo Castrillon. TeIL: a type-safe imperative Tensor Intermediate Language. In Proceedings of the 6th ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming (ARRAY), ARRAY 2019, pages 57–68, New York, NY, USA, June 2019. ACM.

[10] Norman A. Rink, Immo Huismann, Adilla Susungi, Jeronimo Castrillon, Jörg Stiller, Jochen Fröhlich, and Claude Tadonki. Cfdlang: High-level code generation for high-order methods in fluid dynamics. In Proceedings of the 3rd International Workshop on Real World Domain Specific Languages (RWDSL 2018), RWDSL2018, pages 5:1–5:10, New York, NY, USA, February 2018. ACM.

[11] William C Skamarock, Joseph B Klemp, Jimy Dudhia, David O Gill, Zhiquan Liu, Judith Berner, Wei Wang, Jordan G Powers, Michael G Duda, Dale M Barker, et al. A description of the advanced research wrf model version 4. National Center for Atmospheric Research: Boulder, CO, USA, 145:145, 2019.

[12] Adilla Susungi, Norman A Rink, Albert Cohen, Jeronimo Castrillon, and Claude Tadonki. Meta-programming for cross-domain tensor optimizations. ACM SIGPLAN Notices, 53(9):79–92, 2018.