http://www.everest-h2020.eu

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

EVEREST

D2.2 – 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	36 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.	D2.2
Deliverable Title	Definition of Language Requirements
Lead Beneficiary	TUD
Type of Deliverable	Report
Dissemination Level	Public
Due date	31/03/2021

Document Information

Delivery date	20/04/2021
No. pages	38
Version | Status	0.5 | final
Responsible Person	Jeronimo Castrillon (TUD)
Authors	Jeronimo Castrillon (TUD), Felix Wittwer (TUD), Karl Friebel (TUD), Gerald Hempel (TUD), Jan Martinovič (IT4I), Stanislav Böhm (IT4I), Martin Šurkovský (IT4I), Michele Paolino (VOS), Fabrizio Ferrandi (PDM), Serena Curzel (PDM), Michele Fiorito (PDM), Christian Pilato (PDM), Stephanie Soldavini (PDM), Gianluca Palermo (PDM), Dionysios Diamantopoulos (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
12/02/2021	0.0	Jeronimo Castrillon (TUD)	Initial draft
11/02/2021	0.1	Felix Wittwer (TUD)	Table of contents
12/02/2021	0.2	Jeronimo Castrillon (TUD)	Rephrasing
23/03/2021	0.3	Felix Wittwer, Gerald Hempel, Karl Friebel, Jeronimo Castrillon (TUD)	First complete draft
29/03/2021	0.4	Jeronimo Castrillon (TUD), Jan Martinovič (IT4I), Stanislav Böhm (IT4I), Martin Šurkovský (IT4I), Fabrizio Ferrandi (PDM), Gianluca Palermo (PDM) and others	Added tables for use cases and tooling requirements
09/04/2021 15/04/2021 16/04/2021	0.5	Jeronimo Castrillon (TUD) Dionysios Diamantopoulos (IBM) Fabrizio Ferrandi (PDM), Christian Pilato (PDM)	Final pass before first complete draft Added introduction to HW design
15/06/2022	0.6	Jeronimo Castrillon (TUD)	Revision according to the comments by the reviewers: Improved conclusions.
11/07.2022	0.7	Jeronimo Castrillon (TUD)	Cleaned up version for resubmission

Quality Control

Approved by internal reviewer	20/04/2021
Approved by WP leader	20/04/2021
Approved by Scientific Coordinator	20/04/2021
Approved by Project Coordinator	20/04/2021
Revision approved by Sc. Coordinator	12/07/2022
Revision approved by Project Coordinator	12/07/2022

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 reports 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 D2.1 and the data requirements formulated in D2.3, they define the work on the EVEREST design environment to be done in work packages 3-6. Therefore, the three deliverables D2.1, D2.2, and D2.3 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 two or all of the deliverables D2.1-D2.3, because 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.

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 Sections 4 and 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 EVERETS. Extra-functional requirements are detailed in Section 7. Finally, Section 8 concludes with a detailed requirement analysis on the different components of the EVEREST programming framework.

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),
D2.5 - Refined definition of language requirements (M24),
D2.6 – Refined definition of data requirements (M24)

2 Introduction

The EVEREST project will put forward 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 development tools that are widely platform agnostic and are able to perform meaningful optimizations on the code poses a unique challenge. These tools should seamlessly integrate in 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 cases 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 with the aim of identifying 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. All use cases are at different levels of maturity, as is explained in detail in the aforementioned document.

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.

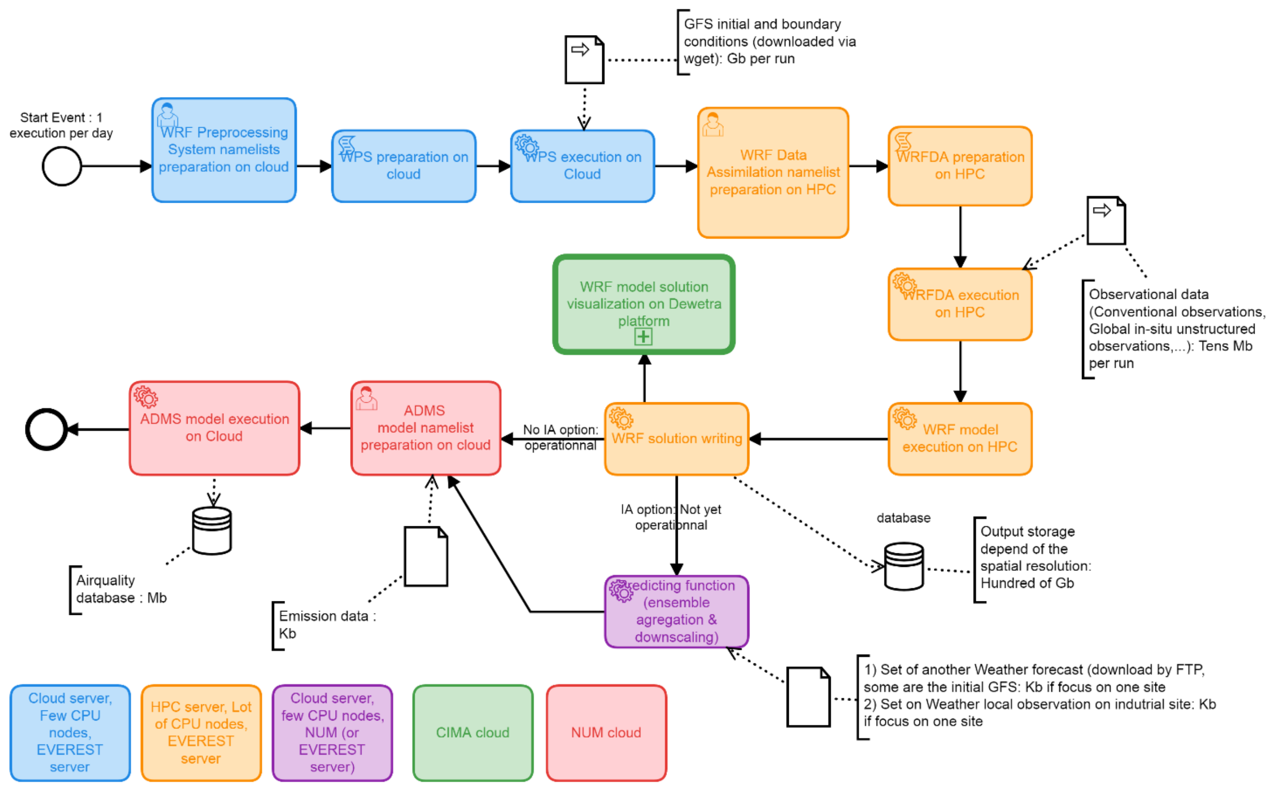
GFS initial and boundary conditions (downloaded via wget): Gb per run
Start Event : 1 execution per day
WRF Preprocessing System namelists preparation on cloud
WPS preparation on cloud
WPS execution on Cloud
WRF Data Assimilation namelist preparation on HPC
WRFDA preparation on HPC
WRF model solution visualization on Dewetra platform
WRFDA execution on HPC
Observational data (Conventional observations, Global in-situ unstructured observations,...): Tens Mb per run
ADMS model execution on Cloud
ADMS model namelist preparation on cloud
No IA option: operationnal
WRF solution writing
WRF model execution on HPC
IA option: Not yet operationnal
database
Output storage depend of the spatial resolution: Hundred of Gb
Airquality database : Mb
Emission data : Kb
Predicting function (ensemble agregation & downscaling)
1) Set of another Weather forecast (download by FTP, some are the initial GFS: Kb if focus on one site
2) Set on Weather local observation on indutrial site: Kb if focus on one site
Cloud server, Few CPU nodes, EVEREST server
HPC server, Lot of CPU nodes, EVEREST server
Cloud server, few CPU nodes, NUM (or EVEREST server)
CIMA cloud
NUM cloud

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 different 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 Domain Specific Languages (DSLs) for computational fluid dynamics, 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 focus of this use case is to optimize traffic flows within cities to reduce congestions and travel times, which in turn can help reducing the pollution caused by traffic. Based on both historical and real-time traffic data as well as weather 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. Both the traffic simulation as well as the traffic prediction model contain kernels that will also 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. Moreover, traffic modeling includes a prediction model and a traffic simulator. The prediction and simulation access the same data, so that the orchestration should allow data sharing between workflows to improve memory efficiency.

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 Orchestration

The use cases are distributed and thus require support for orchestration among the different workflows. We will base the discussion in this section on the traffic simulation use case. A similar analysis applies to the other use cases. In the particular case of traffic simulation, additional language support maybe required to optimize the workflow as discussed in the following.

4.1 Problem Description

As outlined in Figure 2, the traffic simulation use case is comprised of four separate workflows that interoperate with each other. A big data collection workflow feeds the traffic simulator, which in turn produces output used for the training of the traffic model and the routing algorithm. Each of these workflows is computationally intensive. Additionally, loads may vary, depending on the input size. Thus, efficient scheduling and orchestration at runtime is necessary.

For the collection and processing of large amounts of data, different standard solutions already exist, making the development of tooling for speeding up this workflow uncritical. Once a solution is chosen, its potential for optimization may be investigated in the future.

Both the traffic simulation and the training of the traffic prediction model are batch operations, being only executed in specific time intervals. HyperLoom [1], a platform for defining and executing workflow pipelines in large-scale distributed environments, will be used to orchestrate these workflows. HyperLoom offers a user-friendly Python interface that eases programmability and already supports batch operations with task-graph model across distributed nodes. Stemming from an EVEREST partner, HyperLoom will continue to be extended to better cater for the EVEREST use cases.

The routing workflow is the most time-critical component of this use case. Continuously processing a stream of incoming routing requests, its need for computing power may wildly vary throughout the day. Due to the streaming-based nature of this aspect of the use case, HyperLoom in its current form is ill-suited, requiring a different tool for the orchestration of this flow. We will analyze a solution with a message queuing system for processing the routing requests and Kubernetes for dynamic resource allocation to adapt to the varying workload.

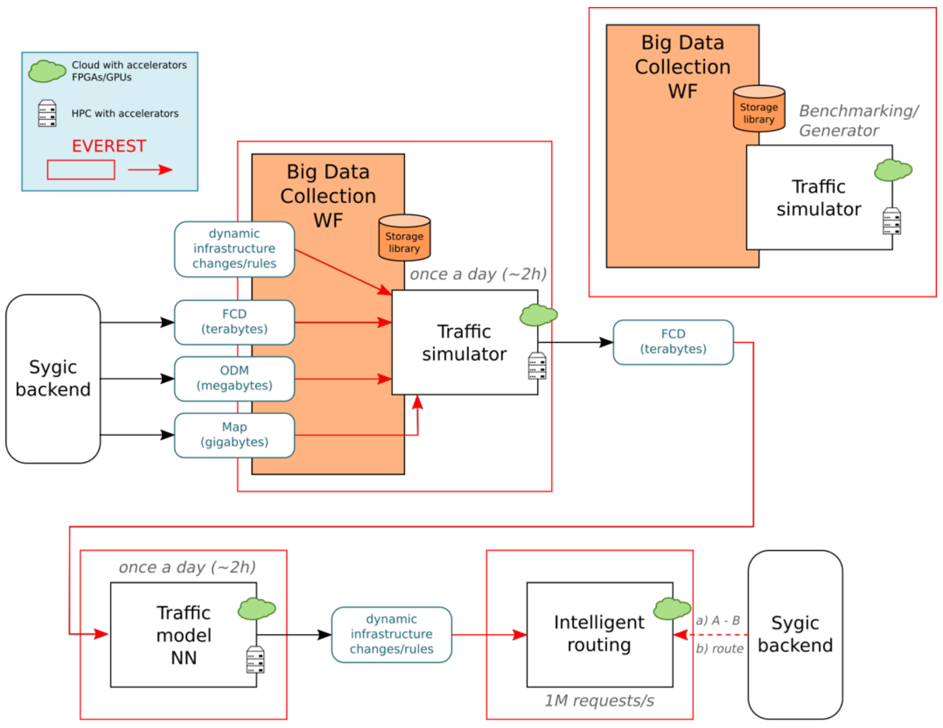

Figure 2: Outline of the traffic simulation use case

4.2 Requirements

Though already a sophisticated platform for describing and executing batch workflows, HyperLoom still needs to be extended for use in this project. Data sharing within and across workflows is a concern that should be realized in this use case to aid computation. This may require HyperLoom to gain a deeper understanding of the data flow within applications. Previous research by project partners can be leveraged for this: Ohua [2], a framework for implicit deterministic parallelism based on extracting and optimizing dataflow graphs and Reactors [3]–[5], a deterministic actor model, based on dataflow extended with discrete event semantics. While well-suited for expressing dataflow-based applications, these two solutions operate at different granularity levels. HyperLoom operates on entire applications, while Ohua and Reactors do it at the level of functions within applications.

To better handle the dynamic nature of the applications running on it, the HyperLoom platform needs to support communicating with the lower layers of the system like the virtualization infrastructure. For the streaming-based routing workflow, a framework needs to be found or devised to handle the highly dynamic nature of the load. Since energy efficiency is a key goal of EVEREST, unneeded computing resources should be freed and made usable for other processes. The automatic adaptation of the runtime could employ solutions like the mARGOt auto-tuner [6].

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 such computations is for solving differential equations using numerical methods. In the case of weather simulations such as WRF, these mainly include fluid dynamics problems. Apart from them, models of microphysical processes to simulate radiation are particularly computationally intensive. For the traffic simulation use case, particle-based simulations are foreseen. At the time of writing it is unclear what kind of particle interactions must be supported with languages. This aspect will thus not be further discussed in this deliverable. 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 can be interpreted as subsets of general tensor algebra. This view is often closer to their actual physical or mathematical formulation. 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. The use cases can thus 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, as they provide our most common form of expert knowledge. Insights gained from the development of optimizations for WRF kernels will later also be applied to kernel computations that are part of the traffic simulation use case and machine learning algorithms. Tensor optimizations for the latter are widespread in the literature.

5.1 Problem Description

The Weather Research and Forecasting Model [7] 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 a simulation missing 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.

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 [8], [9]. 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 have to be extended to cater for the stencils in the WRF model and the compiler must be retargeted to generate code for the EVEREST platform. Most notably, the compiler must generate code 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).

5.2.1 Contextual Requirements

Standalone DSLs can provide great performance improvements. In reality, DSLs have to 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) 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 must be aware of the immediate surrounding context, such as data types, data layout and functional conventions. The DSL and its compiler must encode just enough information to enable integration without cluttering the design.

- The user, or preferably the toolchain, must augment the existing codebase with a customization point that allows invoking the DSL implementation. In other words, both the tooling and the language must accommodate their respective parts of a (foreign) calling mechanism.
- 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 language 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.
- The DSL must map to an intermediate representation that makes it amenable to optimizations within the abstract target domain, which should include mathematical transformations along with more hardware-specific ones at a lower abstraction level.
- The chosen representation must either build on or enable the use of existing or standardized infrastructure so that the toolchain can target as many platforms as possible and interoperate with other tools in the EVEREST project.

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.

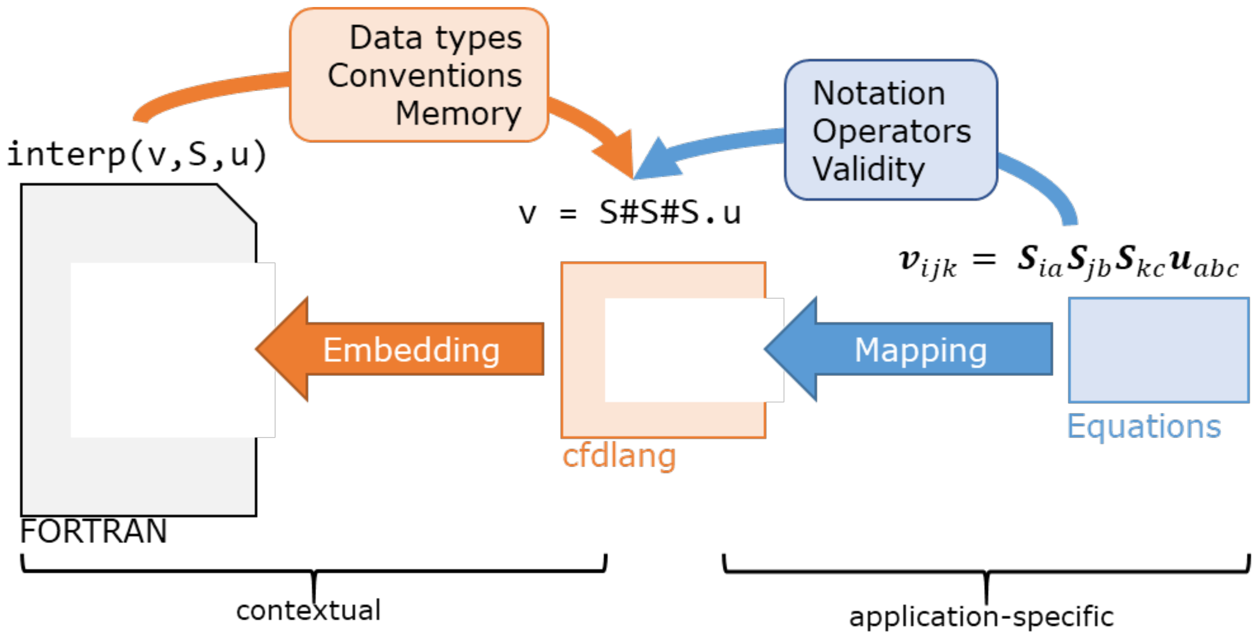

Figure 3: Requirement factors for the Kernel DSL

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 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.

6 Hardware Design Considerations

6.1 HLS problem description

Field Programmable Gate Arrays (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, the common approach in using FPGAs to accelerate complex applications relies on High-Level Synthesis (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 makes the most time-consuming step in the development flow automatic. 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 generated by HLS usually comes with standard interfaces making possible the integration of the accelerators in more complex system-on-chip architectures. In EVEREST, the interfaces between the accelerators and the rest of the platform (see 6.3) will be based on the Advanced eXtensible Interface (AXI), part of the ARM Advanced Microcontroller Bus Architecture specifications.

6.2 HLS challenges

Current HLS tools effectively generate serial or parallel (e.g., through OpenCL annotations) accelerators for regular, easily partitionable, arithmetic-intensive workloads typical of digital signal processing. They mainly target extraction of instruction-level parallelism and consider simple memory subsystems. Additionally, they typically do not consider the need to operate with large datasets that cannot fit into on-chip memories or cannot be localized. Thus, they consider known, fixed memory access latencies and perform optimizations that reduce such latencies. In general, they do not consider massive but fine-grained memory parallelism due to datasets that can barely fit in the external accelerator memory, the data-dependent operations, the highly unbalanced parallel activities, the synchronization through atomic memory operations, and the creation of custom memory architectures around the computational logic.

HLS-based solutions for optimizing the memory accesses through the exploitation of coarse-grained parallelism will make the EVEREST approach amenable to the power efficient execution of workload with large datasets. In addition, accelerators will become more efficient with the proper optimization of memory accesses and data transfers by using intelligent memory managers, automatically generated with a combination of compiler information and hardware generators.

On the same front, EVEREST will extend the support for floating-point computation with variable precision to further improve energy and latency savings. For example, in Machine Learning/AI-based tasks, such variable precision operations will reflect different quantization 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 6. 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 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.

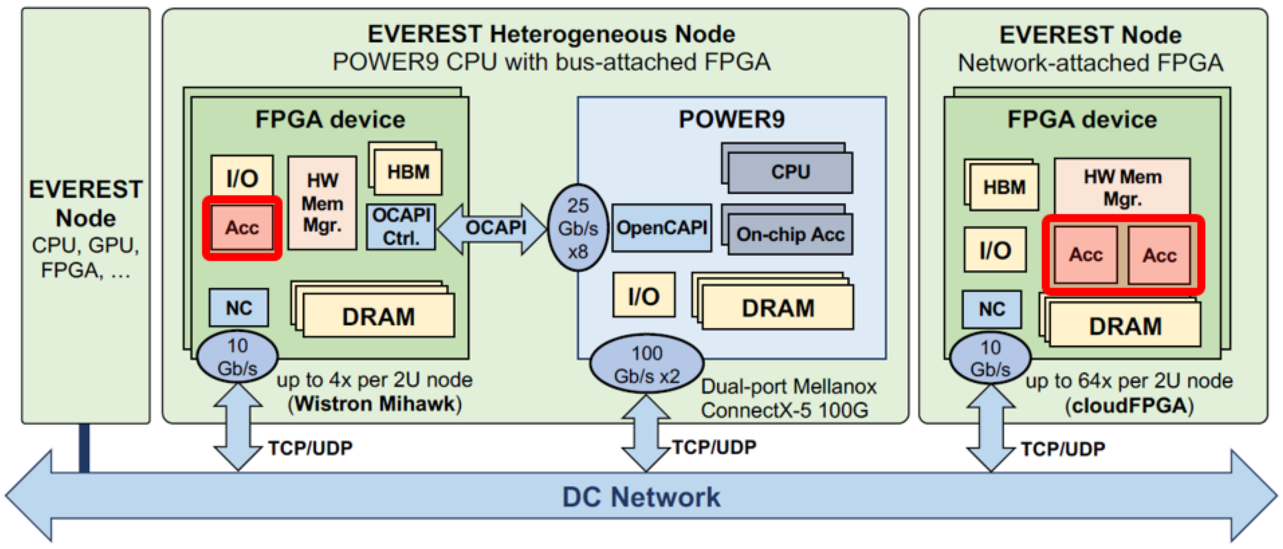

Figure 4: EVEREST experimental heterogeneous platforms

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 RTL-level. Both interfaces are offered by the Integrated Development Environments (IDEs) of the two platforms, i.e., the OC-Accel framework of the POWER9 with the OpenCAPI-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 OC-Accel and cFDK provide a Memory Mapped I/O register access over an AXILite bus, as well as a full AXI master bus. cFDK also enables AXI-stream based access. Same AXI master buses are used to connect the accelerators to the FPGA DRAM channels (also HBM for OC-Accel).

The OC-Accel logic and the cFDK logic implement all the necessary low-level processing of the OpenCAPI 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 High-Level-Synthesis (HLS) tools with *#pragma* directives at the function definition level.

On the host software side, the two platforms offer different APIs. OC-Accel relies on the libcxl user-space and the ocxl kernel-space libraries. 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.

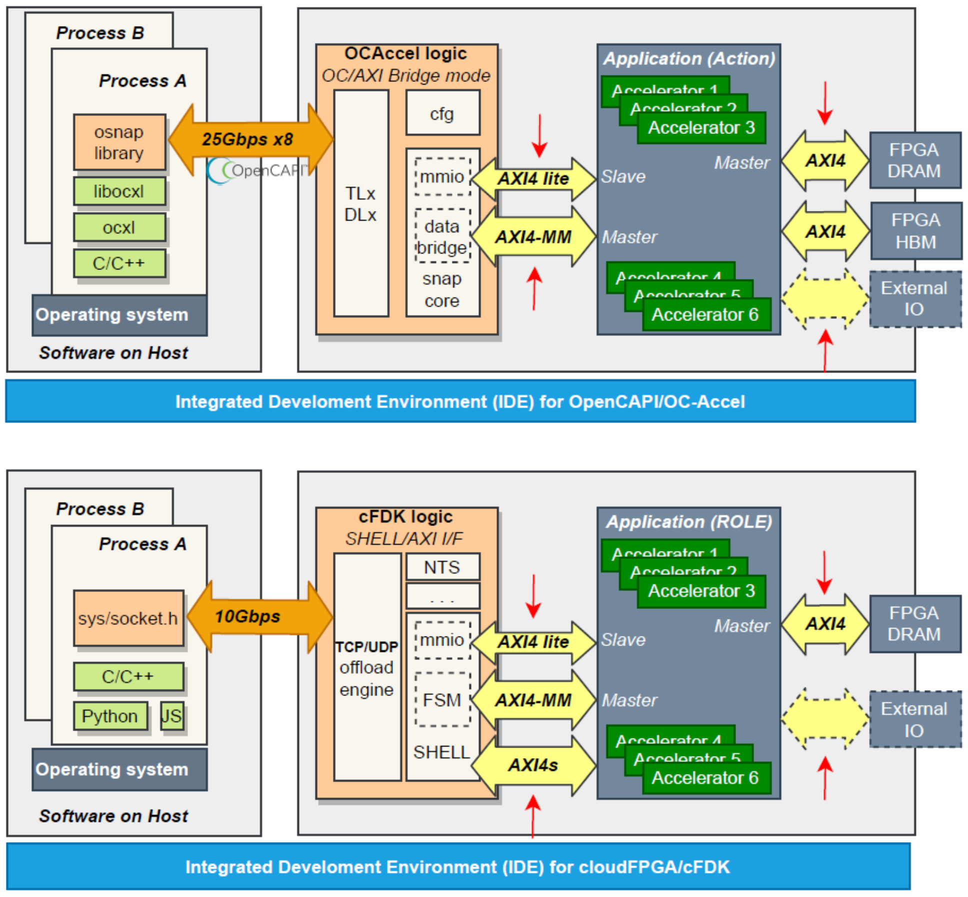

Figure 5: Interface requirements for FPGA accelerators a) at the host software side and b) at the FPGA side for both OC-Accel (top) and cFDK (bottom)

7 Extra-Functional Information

While language and compiler design is mainly focused on meeting functional requirements, it also has to ensure that extra-functional requirements are met. This requires additional information to be passed on from the input code to the final program. Extra-functional information could be provided in the form of compiler flags, annotations within the code or using an additional markup language.

This section briefly discusses extra-functional requirements identified during the use case analysis.

7.1 Timing Constraints

Some applications need to analyze large and varying amounts of data and still have to deliver results within a specific timeframe or else the data loses its value. This is especially true for the weather simulation use cases, where weather predictions have to be produced so that the entity requesting the information has time to act based on the results. The traffic simulation use case is even more time critical as the simulation of the current traffic situation has to keep up with real-world developments to deliver accurate navigation information.

These timing constraints have to be considered by the EVEREST tool flow and hence must be representable in the language abstractions and the compiler. End-users must be able to annotate timing constraints to certain computations that are then enforced by the EVEREST platform. This enforcement could for instance be realized either by aborting computations that would cause the overall execution to miss the deadline, allocating more computing resources if possible or by reducing the accuracy of certain computations in favor of meeting the deadline.

7.2 Energy Efficiency

HPC clusters need non-negligible amounts of energy to operate. Apart from high-performance and respecting timing constraints, the EVEREST platform seeks to also reduce the energy footprint of the applications. Heavy and efficient use of FPGA acceleration will help making systems more energy efficient. The compiler and the HLS flows will use energy as additional metric to drive optimization. This metric is however more difficult to estimate and not trivial to measure accurately. We expect to be able to rely on advanced measuring setups (for instance at TU Dresden's super computer) to shed light on energy-aware optimizations for the kinds of large scale application dealt with in EVEREST.

7.3 Data Security

Some of the data that is processed as part of the different application use cases may be considered confidential and must therefore be specifically protected from unauthorized access. Measures must be available to ensure the integrity and availability of the data. The EVEREST tool flow will have to account for these requirements. A detailed analysis of the requirements on data security is provided in Deliverable D2.3.

8 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.

8.1 Summary: Properties of the Use Cases for Programming Support

Table 1 High-level properties of EVEREST use cases 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 HyperLoom). 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.

Table 1 High-level properties of EVEREST use cases

	Distrib-uted	Multi-location	Multi-language	Streaming	Batch	Time criticality	ML compo-nents	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

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 selected components are shown in Table 2. These components are selected for being critical for the execution of the use cases. 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.

Table 2 Properties of key components in EVEREST use cases

ID	**Name**	**Languages**	**Target resource**	**Compute class**	**Data class**	**Frameworks**
C1	WRF Assimilation	Fortran	CPU, GPU, FPGA	HPC, I/O bound	Regular data	WRF
C2	WRF data preparation	Fortran	CPU, FPGA	HPC, I/O & Memory bound?	Regular data	WRF
C3	WRF radiation	Fortran	CPU, GPU, FPGA	HPC, compute bound	Regular data	WRF
C4	WRF cloud movement and microphysics	Fortran	CPU, GPU, FPGA	HPC, compute bound	Regular data	WRF
C5	Energy modeling	Python, C++			Irregular data	TF, Keras
C6	Big data collection	Rust/Python	CPU, FPGA	I/O and storage bound	Irregular data	Expected: Sqlite, HDF5, InfluxDB, Apache Flink
C7	Traffic: AI Training	Python	CPU, GPU, FPGA	HPC, Compute intensive	Regular data	TF, Keras
C8	Traffic: AI inference	Python	CPU, GPU, FPGA	Cloud, Edge	Regular data	TF, Keras
C9	Traffic simulation: Benchmarking & AI training	Python, Rust, C++	CPU, GPU, FPGA	HPC, Compute intensive	Irregular data	HyperLoom
C10	Traffic simulation: daily use	Python, Rust, C++	CPU, GPU, FPGA	HPC, Cloud Edge	Irregular data	HyperLoom
C11	Intelligent routing	C++, Rust	CPU, GPU, FPGA	HPC, Cloud	Irregular data	Unknown

8.2 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 Global requirements on key use case components (the darker, the more important the requirement is). The table includes the following requirements:

GREQ1. Programmability: End users of the platform should profit from the EVEREST platform in a transparent way. 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 HDL. Code modifications include annotations or inserting DSL expressions in exiting 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) do not require that much programming support, since this is already accounted for in machine learning frameworks.

GREQ2. Interoperability: End programmers use different frameworks (cf. Table 2) and languages. A learned model, for instance, has to 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.

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 second objective.

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

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	░	░	▓	▓	▓	░	░	▓	░	▓	▓

From these global requirements and the information in Table 1 High-level properties of EVEREST use cases and Table 2, we distil requirements for different flows to be implemented in the EVEREST software development kit. The flows and the requirements are described hereafter.

8.2.1 Overall Envisioned Flow

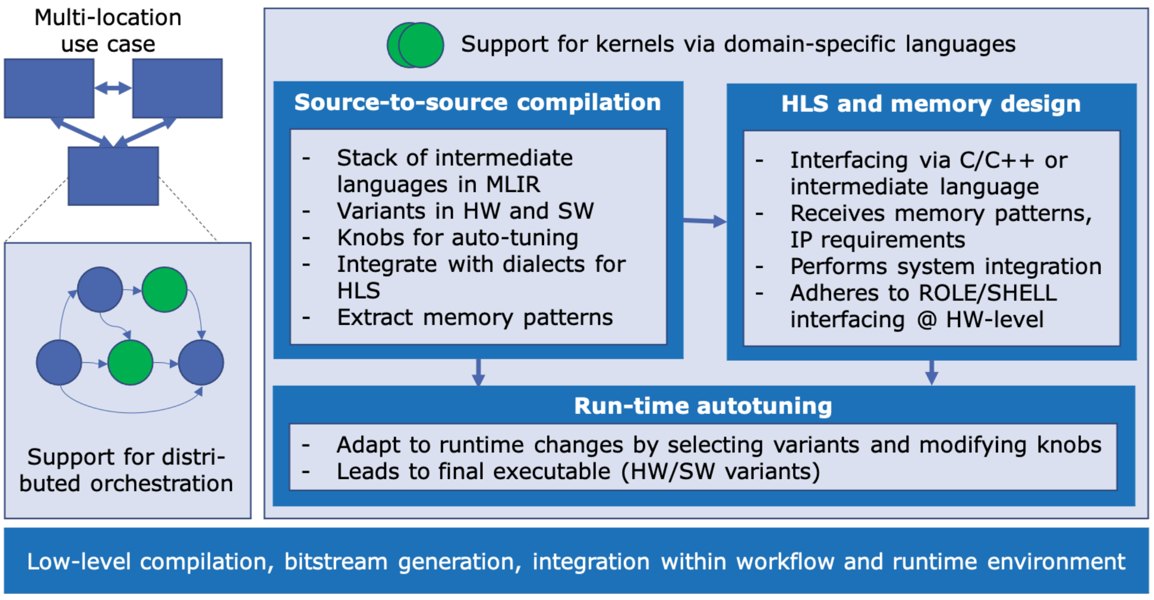
Multi-location use case
Support for distri-buted orchestration
Support for kernels via domain-specific languages
Source-to-source compilation
- Stack of intermediate languages in MLIR
- Variants in HW and SW
- Knobs for auto-tuning
- Integrate with dialects for HLS
- Extract memory patterns
HLS and memory design
- Interfacing via C/C++ or intermediate language
- Receives memory patterns, IP requirements
- Performs system integration
- Adheres to ROLE/SHELL interfacing @ HW-level
Run-time autotuning
- Adapt to runtime changes by selecting variants and modifying knobs
- Leads to final executable (HW/SW variants)
Low-level compilation, bitstream generation, integration within workflow and runtime environment

Figure 6. Envisioned overall 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 Sections 8.2.2 through 8.2.6.

8.2.1.1 Workflow Orchestration

The preliminary use case analysis in Section 3 showed that some use cases include highly computing intensive workflows. Within the project, HyperLoom [1] shall be used to orchestrate these large application flows. HyperLoom 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 HyperLoom server that distributes the work onto several workers.

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.

8.2.1.2 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 computational fluid dynamics and tensor-based computation, e.g., CFDLang [2], TeML [9], and TeIL [10]. 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.

8.2.1.3 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 [11]. 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[1]). 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.

8.2.1.4 High-level Synthesis and Memory Design

High-level synthesis (HLS) allows application designers to accelerate specific kernels on FPGA without having much hardware/software. 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, 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 [12]. Supporting two different HLS tools shows the interoperability of our solutions. Xilinx Vitis HLS is one of most common HLS tools. It offers an open-source frontend based on the LLVM compiler. It also supports a wide range of optimization directives and accelerator interfaces. Bambu accepts as input standard C/C++ specifications, OpenMP parallel annotations, 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 HLS tool already includes many hardware-oriented optimizations and interfaces with synthesis and verification backends, either commercial or open-source. So, it is also a good platform to evaluate targets different from Xilinx devices.

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 [13], 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.

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

8.2.1.5 Run-time Auto-Tuning

As outlined in Section 7.1 , the timely execution of parts of the use case application is of utmost importance. However, 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 [6]. 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 like Computational Fluid Dynamics, mARGOt could be used to regulate the execution frequency of computationally intensive kernels or to trade accuracy in the results for higher throughputs, e.g., by reducing the polynomial degree of interpolation operations if necessary.

8.2.2 Requirements: Orchestration Large Application Flows (DAGs)

Table 4 Requirements for EVEREST HyperLoom extensions

ID	**Name**	**Description**	**Nature**	**Priority**	**Comments**	**Relation to global**
REQ2.1	Front-end for EVEREST Applications	Framework on top of HyperLoom for easy use-cases driven development	Tool	Must have	Specific front-end design by use-cases requirements	GREQ1
REQ2.2	Dynamic data sharing between DAG tasks	Extend framework to support spawning a dynamic service-like tasks that may serve data independently on fixed dependencies defined in a task graph	Methodology/ API	Should have	In some frameworks it is known as actor model (e.g. in Ray).	GREQ4
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

8.2.3 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)	Methodology	Must have	Kernel support for WRF simulations.	GREQ1
REQ3.2	WRF Fortran integration	Expression abstractions should be callable from within Fortran code	Methodology	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	Methodology/ API	Should have	Allow importing models to hook into the code generation process for EVEREST specific transformations	GREQ2, GREQ4
REQ3.4	Streaming support	Language support for streaming workflows with highly dynamic loads	Methodology	Could have	Enable compiler reasoning for reconfiguring streaming oriented computations. Expected to support traffic use case. Will be revisited as the implementation fort 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)	Methodology/ tool	Should have	By contributing to open sources 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	Methodology	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	Methodology/ tool	Could have	This should extend on previous work on optimization for dataflow programs (including mapping, graph rewrites and I/O batching)	GREQ4, GREQ5
REQ3.8	Multi-target code generation	The source to source compiler should generate code for different targets	Methodology/ 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.	Methodology/ tool	Must have (for adaptable 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	Methodology	Must have	The compiler must export code (or an intermediate representation thereof) to the HLS flow, including behavioral descriptions of the kernels alongside configuration information for generation/configuration of the memory modules	GREQ4, GREQ5
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

8.2.4 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++ Language support for HLS of descriptions coming from DSL compiler	Methodology/tool	Must have	The language supported should covered the original code in case it is written in C/C++ or the one generated by the DSL compiler	GREQ1, GREQ2
REQ4.2	Bambu LLVM bytecode support	Low level integration with DSL compiler	Methodology/Tool	Must have	The version of the LLVM bytecode should be consistent with used by the DSL compiler.	GREQ2, GREQ4
REQ4.3	Bambu MLIR dialect support	Direct synthesis from MLIR dialects	Methodology/tool	Can have	It may improve the final performance raising the abstraction level. At least, it should support the affine dialect.	GREQ2, GREQ4
REQ4.4	HLS Verilog output	HLS generates RTL Verilog code as output	Methodology/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	Methodology/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-along function and needs to be specified	Methodology/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	Methodology/tool	Must have	The start, done, etc. protocol of the top component has to be defined and compatible with the EVEREST platform.	GREQ2
REQ4.8	Port-Level interfaces	IO interface protocols added to the individual function arguments	Methodology/tool	Must have	The definition of protocol should be defined through code annotations	GREQ1, GREQ2
REQ4.9	Bambu Vivado HLS IO interface interoperability	Annotations specifying the IO protocols interface compatibility	Methodology/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.	Methodology/tool	Must have	These technology constraints will passed as input options.	GREQ2
REQ4.11	Bambu Data flow annotations	HLS Data flow support	Methodology/tool	Should/have	Dataflow style applications could be specified by code annotations	GREQ1
REQ4.12	Bambu OpenMP support	OpenMP for pragma synthesis support	Methodology/tool	Must have	The body of OpenMP parallel loop need to be in a separate function. Example of supported code could be found at OpenMP for[2] and OpenMP simd[3]	GREQ1
REQ4.13	Bambu floating point precision	Floating point variables may use a custom floating precision data type.	Methodology/tool	Can have	Allow optimizations of scientific and machine learning kernels.	GREQ1

[2] https://github.com/ferrandi/PandA-bambu/blob/main/examples/parallel_queries/trinityq1/lubm_trinityq1.c
[3]https://github.com/ferrandi/PandA-bambu/blob/main/examples/omp_simd/add/modified/add.c

REQ4.14	cFDK/OC-Accel top component interface	Interface definition of the top component being intergraded with cFDK/OC-Accel frameworks.	Methodology/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	Methodology/tool	Must have	The HLS-generated kernels and the memory modules should have a common interface format	GREQ2
REQ4.16	Software-level support	Software code to interface with the accelerators.	Methodology/tool	Must have	The accelerators should be invoked with custom OS drivers	GREQ1
REQ4.17	Hardware/software data sharing	Data allocation must be compatible with hardware memory interfaces	Methodology/tool	Must have	The software-level data allocation should be performed in a way that hardware can access the data	GREQ2

8.2.5 Requirements: Autotuning and Virtualized Environment

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

ID	**Requirement**	**Description**	**Nature**	**Priority**	**Comments**	**Relation to global**
REQ5.1	Application knobs	The autotuning framework should have access to the application knobs	Methodology/ 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	Methodology/ 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	Methodology/ tool	Could have	Given the knowledge of the application, the dynamic adaptation framework can	GREQ2

		with the virtualized environment			provide hints to the virtual manager to steer decisions	
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	Methodology/ 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	Methodology/ 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	Methodology/ 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	Methodology/ 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	Methodology/ 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
REQ5.9	HW monitors	HW accelerators should expose a monitor interface to the run-time to trigger dynamic decisions	Methodology/ 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	Methodology/ 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	Methodology / tool	Must have	The EVEREST virtualization environment targets different hardware accelerators and CPU architectures	GREQ2, GREQ4, GREQ5

8.2.6 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.

9 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 hardware design of the EVEREST platform and listed extra-functional information from the use cases that has to be accounted for in the EVEREST SDK. 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. On top of this, the use cases are not fully specified but are meant to evolve throughout the course of the project. Provided our current understanding of the use cases, these are the main findings that will drive the development of the compilation framework in the EVEREST SDK:

- A need for interoperability for machine learning algorithms and optimization for distributed execution of large models on FPGAs.
- A need for high-level abstractions in HPC simulations to transparently optimize hardware, with focus on memory subsystems and data movement.
- A need to implicitly describe task graphs in a syntax close to application developers (e.g., Rust or Python).

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 enable 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 routing use case which is well understood at the time this deliverable was written.

Apart from these requirements, 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 will serve as guide for the work in work packages WP4 and WP5.

10 References

[1]V. Cima *et al.*, "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 - PARMA-DITAM '18*, Manchester, United Kingdom, 2018, pp. 1–6, doi: 10.1145/3183767.3183768.

[2]S. Ertel, C. Fetzer, and P. Felber, "Ohua: Implicit Dataflow Programming for Concurrent Systems," in *Proceedings of the Principles and Practices of Programming on The Java Platform*, Melbourne FL USA, Sep. 2015, pp. 51–64, doi: 10.1145/2807426.2807431.

[3]M. Lohstroh and E. A. Lee, "Deterministic Actors," in *2019 Forum for Specification and Design Languages (FDL)*, Southampton, United Kingdom, Sep. 2019, pp. 1–8, doi: 10.1109/FDL.2019.8876922.

[4]M. Lohstroh, C. Menard, A. Schulz-Rosengarten, M. Weber, J. Castrillon, and E. A. Lee, "A Language for Deterministic Coordination Across Multiple Timelines," in *2020 Forum for Specification and Design Languages (FDL)*, Kiel, Germany, Sep. 2020, pp. 1–8, doi: 10.1109/FDL50818.2020.9232939.

[5]M. Lohstroh *et al.*, "Reactors: A Deterministic Model for Composable Reactive Systems," in *Cyber Physical Systems. Model-Based Design*, vol. 11971, R. Chamberlain, M. Edin Grimheden, and W. Taha, Eds. Cham: Springer International Publishing, 2020, pp. 59–85.

[6]D. Gadioli, E. Vitali, G. Palermo, and C. Silvano, "mARGOt: A Dynamic Autotuning Framework for Self-Aware Approximate Computing," *IEEE Trans. Comput.*, vol. 68, no. 5, pp. 713–728, May 2019, doi: 10.1109/TC.2018.2883597.

[7]W. C. Skamarock *et al.*, "A Description of the Advanced Research WRF Model Version 4," UCAR/NCAR, Mar. 2019. doi: 10.5065/1DFH-6P97.

[8]N. A. Rink, A. Susungi, J. Castrillon, J. Stiller, and C. Tadonki, "CFDlang: High-level code generation for high-order methods in fluid dynamics," in *Proceedings of the Real World Domain Specific Languages Workshop 2018 on - RWDSL2018*, Vienna, Austria, 2018, pp. 1–10, doi: 10.1145/3183895.3183900.

[9]A. Susungi, N. A. Rink, A. Cohen, J. Castrillon, and C. Tadonki, "Meta-programming for cross-domain tensor optimizations," in *Proceedings of the 17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, Boston MA USA, Nov. 2018, pp. 79–92, doi: 10.1145/3278122.3278131.

[10] N. A. Rink and J. 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 2019*, Phoenix, AZ, USA, 2019, pp. 57–68, doi: 10.1145/3315454.3329959.

[11] C. Lattner *et al.*, "MLIR: A Compiler Infrastructure for the End of Moore's Law," *ArXiv200211054 Cs*, Feb. 2020, Accessed: Mar. 26, 2021. [Online]. Available: http://arxiv.org/abs/2002.11054.

[12] F. Ferrandi *et al.* "Invited: Bambu: An Open-Source Research Framework for the High-Level Synthesis of Complex Applications", Proceedings of the ACM/IEEE Design Automation Conference (DAC), 2021.

[13] C. Pilato et al. "System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip" in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2017.