A System Development Kit for Big Data Applications on FPGA-based Clusters: The EVEREST Approach

Christian Pilato[†], Subhadeep Banik[‡], Jakub Beránek[‖], Fabien Brocheton[‡‡], Jeronimo Castrillon[§], Riccardo Cevasco[††], Radim Cmar[x], Serena Curzel[†], Fabrizio Ferrandi[†], Karl F. A. Friebel[§], Antonella Galizia[¶xi], Matteo Grasso[††], Paulo Silva[‖], Jan Martinovic[‖], Gianluca Palermo[†], Michele Paolino[**], Andrea Parodi[¶], Antonio Parodi[¶], Fabio Pintus[¶], Raphael Polig[*], David Poulet[‡‡], Francesco Regazzoni[xii‡], Burkhard Ringlein[*], Roberto Rocco[†], Katerina Slaninova[‖], Tom Slooff[‡], Stephanie Soldavini[†], Felix Suchert[§], Mattia Tibaldi[†], Beat Weiss[*], Christoph Hagleitner[*]

[*]*IBM Research Europe, Switzerland*, [†]*Politecnico di Milano, Italy*, [‡]*Università della Svizzera italiana, Switzerland*, [§]*Technische Universität Dresden, Germany*, [¶]*Centro Internazionale di Monitoraggio Ambientale, Italy*, [‖]*IT4Innovations, VSB – Technical University of Ostrava, Czech Republic*, [**]*Virtual Open Systems, France*, [††]*Duferco Energia, Italy*, [‡‡]*NUMTECH, France*, [x]*Sygic, Slovakia*, [xi]*IMATI - Consiglio Nazionale Ricerche (CNR), Italy*, [xii]*University of Amsterdam, The Netherlands*

***Abstract*—Modern big data workflows are characterized by computationally intensive kernels. The simulated results are often combined with knowledge extracted from AI models to ultimately support decision-making. These energy-hungry workflows are increasingly executed in data centers with energy-efficient hardware accelerators since FPGAs are well-suited for this task due to their inherent parallelism.**

We present the H2020 project EVEREST, which has developed a system development kit (SDK) to simplify the creation of FPGA-accelerated kernels and manage the execution at runtime through a virtualization environment. This paper describes the main components of the EVEREST SDK and the benefits that can be achieved in our use cases.

I. Introduction

Modern big data applications often require processing large volumes of data with massively parallel algorithms (including machine learning ones). They are used in several application domains, like scientific computing, healthcare and medicine, transportation, etc. Each algorithm is generally described as a *workflow*, which is executed on cloud-based or on-premise infrastructure and offloading critical calculations to hardware accelerators, e.g., FPGAs. Designers face several challenges in generating efficient FPGA accelerators.

First, application designers must write code that the hardware generation tools can understand. Today, application designers generally use a variety of programming languages (e.g., Fortran, Python, Rust, C/C++, domain-specific languages), while hardware generation tools are usually limited to a few of them (e.g., C/C++/SystemC) [13], demanding extensive and error-prone code rewriting. On top of this, selecting and using proper custom data formats is essential to obtain efficient implementations without sacrificing accuracy [12, 24]. Then, depending on the architecture of the target platform, several optimizations can be applied to the FPGA system architecture to improve the execution, especially regarding data management. For example, data layouts in memory must match how the information is read/written to avoid bottlenecks. Unfortunately, existing toolchains do not support automatic optimizations, so hardware experts must hand-craft the solutions, which is tedious and error-prone. For this reason, they often apply standard optimizations based on well-known application patterns (e.g., loop unrolling and pipelining) or system architectures (e.g., double buffering), easily leading to sub-optimal solutions. Finally, the availability of resources at runtime may prevent the system from meeting application requirements. To account for this, applications should adapt to the environment, changing how the computation is performed. To the best of our knowledge, no single framework for big data applications supports all these features.

EVEREST is a H2020 EU project that aims at simplifying the development of complex big data applications for FPGA-based data centers [15]. The EVEREST System Development Kit (SDK) is a framework for optimizing selected kernels in the application workflow. We first introduce our application use cases (Section II) and our target platform prototype (Section III). We then introduce the EVEREST SDK (Section IV), and we detail our main contributions: (1) a data-driven compilation framework (Section V), (2) a virtualized runtime environment (Section VI), and (3) an anomaly detection service (Section VII). In Section VIII, we describe the current EVEREST prototypes and the technical insights that we obtained, while Section IX concludes the paper.

II. Application Use Cases

The development of the EVEREST SDK is driven by three use cases: an application for the prediction of renewable energy production, an application for air-quality monitoring, and a traffic modeling and prediction system. The first two use

979-8-3503-4859-0/DATE24/© 2024 EDAA

cases are based on weather predictions. We now describe the relevant aspects of all these workloads.

A. WRF-based Weather Simulations

Modeling weather scenarios is at the base of many environmental and societal challenges. In EVEREST, we rely on WRF, a state-of-the-art numerical model for weather forecasts [17]. The model can operate at spatial resolutions from hundreds of meters to hundreds of kilometers. WRF also provides the data assimilation system, called WRFDA, since the ingestion of observational data represents valuable support to weather prediction by improving the initial condition of the problem [2]. Weather prediction models are HPC applications with high demands of computational and storage resources.

CIMA exploits WRF daily as a high-resolution limited-area model for meteorological research and operational weather forecasts. In EVEREST, CIMA aims to improve the accuracy of weather predictions by (1) pushing forward data assimilation and achieving better descriptions of the atmospheric state used as WRF initial conditions, (2) speeding up the WRF execution by means of the EVEREST FPGA nodes to implement and test an ensemble prediction. An accelerated WRF implementation can enable new market opportunities for other application sectors (e.g., high-impact weather event prediction, precision agriculture, pedestrian path planning) thanks to the possibility of evaluating the impact of more frequent and possibly more accurate simulations.

B. Renewable-energy prediction

The energy prediction use case concerns the forecast of the power generated by a wind farm. The application target aims to help energy traders reduce forecast errors in the predictions used in short-term markets for the entire renewable portfolio and related unbalanced costs. To do so, the application requires (1) a weather forecast with the evolution of meteorological parameters (e.g., wind speed), updated at hourly resolution, and (2) parameters and historical data of the wind farm (e.g., measured wind speed, availability of the wind turbines and transmission systems. The input of the WRF numerical model has been customized to fit the wind farm site topography and provide the forecast at different height levels to get closer to the wind turbine height. EVEREST uses a machine-learning approach combining deterministic weather forecasts, historical WRF time series, historical datasets of the wind farm, and real-time data, trained with at least one year of data. The current version of the application uses the Kernel Ridge algorithm, which considers wind-related parameters and the corresponding energy produced in the farm.

C. Air-quality monitoring

In the air-quality use case, we aim to forecast the impact of atmospheric releases of an industrial site on its surrounding environment and adapt site activities to avoid pollution peaks. Such application has a two/three-day time window and combines (1) a weather forecast at hourly resolution with (2) an atmospheric air-quality forecast, along with forecasts of the site emissions and some fixed parameters (e.g., the local topography and land use around the site, the site buildings, the emission velocity or temperature) In the case of high impacts, the industrial site can activate emission reduction processes to respect acceptable pollution levels. Such actions have a financial cost (tens of thousands of euros per day), so they should be used only when needed. The industrial site decides to plan its activity for the next days in the morning. So, reducing the time to obtain the forecasts is essential.

The accelerated WRF allows for accurate weather forecasts, while errors are limited with machine learning. The ML-based method will combine multiple weather forecasts (due to the natural uncertainties of numerical weather simulations) forced by local weather observations on-site. The approach focuses on three weather parameters that are frequently observed: the air temperature at 10m, the wind direction, and the wind speed.

D. Traffic modeling

Precise traffic models and predictions help understand and improve every-minute traffic conditions. Our use case is a *traffic ecosystem* for precise and fast calculations of traffic model and prediction, both short-term (e.g., a subsequent hour) and long-term (e.g., any given future time point). We use (a) floating car data (FCD) (from mobile devices used in Sygic navigation) that define vehicle speeds on GPS positions across the road network; (b) origin-destination matrix data (ODM) (from mobile operators) that define the overall mobility of citizens across the city grid; (c) meteorological data such as temperature and precipitation. With computationally intensive algorithms, we calculate the traffic model, which is represented by (a) macroscopic parameters for each road segment (speed, flow, intensity) for each 15-minute interval over a weekday and (b) coefficients of the prediction model for each road segment. On top of the models, we build traffic prediction and intelligent routing. The *traffic ecosystem* regularly updates its model with new daily incoming data. To improve the system quality, we use (1) a convolutional neural network for training the road speed prediction model; (2) a Hidden Markov model for map matching of sparse and noisy FCD points on a road network; (3) a Gaussian Mixture model for an alternative traffic prediction with incomplete data; (4) Probabilistic Time Dependent Routing to infer correct arrival times.

The *traffic ecosystem* poses challenges both in storage and processing. Big data sets must be transferred across components, stored, and available for processing. Computations must be fast and cost-efficient to meet a daily processing cycle.

III. EVEREST Target Systems

The EVEREST target system is a *converged heterogeneous platform*, as depicted in Figure 1. In the back end, the platform comprises computing nodes with FPGA-based compute accelerators. In the front end, the platform features a unified workflow deployment based on Jupyter Notebooks. In between, a middleware stack of technologies consolidates the hardware capabilities of the back end into consumable services. We consider computing nodes that include IBM Zurich's on-premise

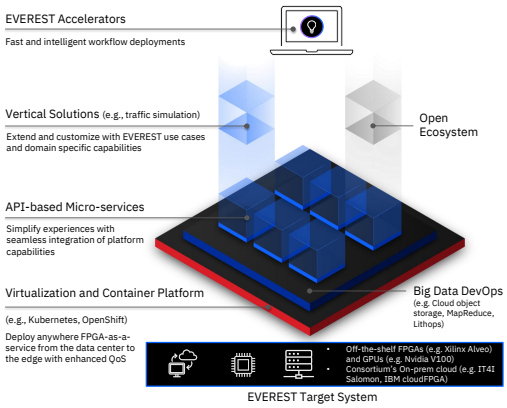

Fig. 1. The EVEREST converged heterogeneous platform

Heterogeneous Compute Cluster and IT4I's on-premise clusters (e.g., Salomon, Barbora, Karolina). The EVEREST target system includes the following architectural components.

EVEREST computing nodes. They include *CPUs* (Intel Xeon, AMD EPYC), *PCIe-attached FPGAs* (AMD Alveo u55c, u280) with Xilinx Runtime (XRT), and *Network-attached FPGAs* (IBM cloudFPGA) directly connected to a 10Gbps TCP/UDP network stack. Both FPGA systems support the creation of complex FPGA system architectures that interface with the rest through standard AXI interfaces. Such systems support both HDL (VHDL, Verilog) and automatic synthesis with high-level synthesis (from C/C++/SystemC).

BigData DevOps. Application developers need a simplified programming interface that enables distributed computing at scale. Many computationally intensive workloads involve large-scale data analytics.

API-based microservices. The evolution of cloud applications into loosely coupled microservices opens new opportunities for hardware accelerators. Components are packaged up in containers as microservices that can handle compute-intensive tasks (e.g., data ingestion, data assimilation, and data processing). Offering such micro-services using RestAPI enables the reuse of the functionality across different use cases.

IV. EVEREST System Development Kit

The EVEREST SDK combines commercial and open-source tools into a unified and interoperable framework to ease the development of complex FPGA system architectures and optimize the runtime execution of the applications. Figure 2 shows an overview of the tools. For example, we can execute high-level synthesis (HLS) with Vitis HLS to generate hardware descriptions for most of the components (including interface protocols) or with Bambu [6] for smooth integration with the compiler and its custom data formats. The EVEREST SDK includes a *data-driven compilation framework*, which creates complex FPGA architectures from high-level descriptions of the kernels to be accelerated, and a *virtualized runtime environment*, which monitors the execution at runtime and adapts it to the surrounding environment. All tools within the SDK are wrapped under the *basecamp* command, which provides a single point of access to the users of the SDK.

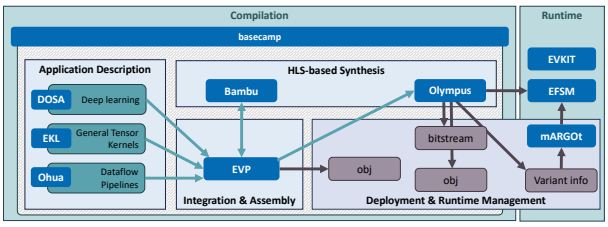

Fig. 2. EVEREST SDK components.

Compilation. The computational kernels marked for FPGA offloading are translated into the corresponding hardware descriptions and, in turn, the bitstream configurations. EVEREST aims to unify the different input languages into a single hardware generation flow based on MLIR and HLS. The compiler backend supports different target platforms (e.g., AMD Alveo and IBM cloudFPGA [20]), also varying the configuration of the memory architectures around the accelerators.

Deployment. The deployment of the application workflows leverages the LEXIS platform[1], which has been extended to offload the execution of selected kernels to FPGA. Once a task (or one of its parts) is marked for FPGA acceleration, its execution is set to be offloaded to FPGA-based clusters.

Execution. During resource allocation, the FPGA cluster may reserve a variable number of nodes for the given application. If the accelerated task requires more resources, the EVEREST runtime can adapt the computation accordingly. Additional dynamic autotuning can be performed to match the characteristics of the data and the required computation to save energy or improve performance.

V. Data-driven Compilation Framework

This section discusses the concrete input languages, intermediate representation, and the hardware generation flow supported by the EVEREST SDK.

A. Input languages and abstractions

The EVEREST SDK leverages high-level domain-specific languages (DSLs) and programming abstractions. These languages can hold rich semantic information to enable system-level architectural exploration, which is hardly possible with the low-level languages supported by mainstream HLS tools. We leverage existing DSLs for physics simulations [9, 22], for dataflow [4], and on popular frameworks for machine learning [3]. As input, the SDK supports standard ONNX ML models, a high-level language for kernels, and a coordination language for high-level dataflow, as detailed below.

1) EVEREST kernel language: To develop the kernel language, we studied the RRTMG radiation module of the WRF code, which consumes around 30% of the compute cycles, For RRTMG acceleration, existing tensor abstractions, e.g., in TVM [3] or CFDlang [22, 23], had to be extended to support in-place construction, broadcasting, index re-association, and subscripted subscripts. The *EVEREST Kernel Language* provides a general syntax for the Einstein Notation [14] as shown

[1]https://lexis-project.eu/web/lexis-platform/

$$\tau_g^M = \sum_{dT} \sum_{dp} \sum_{d\eta} r_{\sigma_x,x,d\eta} \alpha_{\sigma_x,x,dT,dp,d\eta} k_{\overline{T}+dT,\overline{p}+dp,\overline{\eta}+d\eta,g}$$

```
i_strato = select(p[x] <= strato, 1, 0)
i_flav   = bnd_to_flav[i_strato, bnd]
i_T      = [j_T, j_T+1]
i_eta    = [j_eta[i_flav[x], x, p], j_eta[i_flav[x], x, p]+1]
i_p      = [j_p+i_strato, j_p+i_strato+1]
tau_abs  = (r_mix[i_flav[x], x, e]
             * f_major[i_flav[x], x, t, p, e]
             * k_major[i_T[x, t], i_p[x, p], i_eta[x, e], g])
```

Fig. 3. Example of major absorber in the EVEREST Kernel Language.

```
fn match_one(gv: GpsVector, mapcell: MapCell) ->
↪  RoadSpeedVector {
    #[kernel(offloaded = true, multiplicity = [1, 1, 1, 1],
        path = "projection.cpp")]
    let cv: CandiVector = projection(gv, mapcell);

    let t: Trellis = build_trellis(gv, cv, mapcell);
    let rsvbb: RoadSpeedVector = viterbi(t, cv);
    interpolate(rsvbb, mapcell)
}
```

Fig. 4. Example ConDRust syntax for map matching a single element.

in Figure 3. This code snippet corresponds to 200 lines of Fortran code in the original WRF RRTMG implementation.

2) EVEREST coordination language: A coordination language is key to integrating the components constituting a larger application. The EVEREST SDK uses an imperative language based on a subset of the Rust programming language called ConDRust [27]. ConDRust is based on Rust's safe ownership model, offers provable determinism (which greatly eases the design of complex heterogeneous systems), and exposes parallelism [5]. In addition to this, the imperative model of ConDRust makes it easier to migrate applications. This is, for instance, the case of the Map-Matching algorithm of the traffic use case, sketched in ConDRust in Figure 4.

The coordination language connects software and hardware components from the kernel language or an ONNX inference model. As a key added value, the language uses rich types to pass the information to hardware-level interface generation.

B. MLIR representations

The EVEREST SDK relies on the MLIR framework [11] to converge different HPC, Big Data, and ML abstractions for system-level optimizations. We contributed several *dialects*, optimizations, and abstraction *lowerings* to implement several compilation flows. The main dialects and their relations are shown in Figure 5. Other frontends can generate representations to enter the EVEREST SDK, like `torch` and `tosa`. ML applications from TVM can be read into the `jabbah` dialect, currently under development. To converge and optimize the different ML DSLs, we follow the concept of Operation Set Architectures [18]. This level of abstraction, captured by `jabbah`, is also used to optimize the distribution of ML applications [19]. The SDK provides dialects for the frontends of the kernel language (`ekl`), the coordination language (`dfg`), and the legacy CFDlang (`cfdlang`). The dialects `ekl` and `cfdlang` can be lowered to an MLIR implementation of the intermediate tensor language [23] (`teil`) and a new dialect for the Einstein notation (`esn`). These abstractions are used to implement a series of transformations [24].

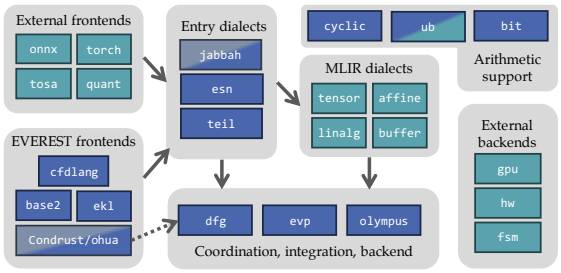

Fig. 5. EVEREST MLIR dialects (in blue) and their integration with other core MLIR dialects (in green). Dialects under construction in grey-blue

Custom data representations are often needed to truly exploit the efficiency of hardware implementations. To this end, the SDK includes a set of dialects to properly model custom data types in MLIR [7], namely, `base2`, `cyclic`, `bit` and `ub`. The latter is currently being moved to core MLIR for proper support for undefined behavior. The remaining dialects handle integration within the EVEREST platform (`evp`) and system-level optimization based on the dataflow of the application (`olympus`).

C. System-level generation

The EVEREST hardware system generation tools, Olympus and DOSA for network attached FPGAs [19], automatically create an optimized FPGA system architecture. Starting with the MLIR description of kernel interactions in the `olympus` dialect [26], the kernel implementations in RTL or C for HLS (for Bambu or Xilinx Vitis), and the FPGA platform details, Olympus creates a custom infrastructure for data movement and organization for the kernels. In the case of network-attached FPGAs, hardware-agnostic synchronous communication routines are generated and inserted [21]. This infrastructure consists of both the necessary hardware modules instantiated on FPGA and host code drivers to move data from host to device and initiate execution on the device. During generation, Olympus will apply relevant optimizations, such as private local memory sharing [16], double-buffering, and read/execute/write pipelining. Additionally, Olympus applies bus-based optimizations such as replicating kernels and dividing a wide memory bus into "lanes" to serve each replication [24], or "packing" the data efficiently to save bandwidth [25].

VI. Virtualized Runtime Environment

A. Resource management

Once the data center has allocated the node resources, the EVEREST resource manager (1) schedules and assigns the workflow tasks to the computational nodes while respecting their dependencies and resource requests; (2) load-balances the computation when necessary; (3) performs data transfers when an input of a task is computed on a different node; (4) monitors the cluster and reschedules tasks if needed.

The runtime interaction with the target applications is done through a Dask-like API, requiring only minimal modifications. The original Dask API is extended with EVEREST-specific features, mainly to specify the resource requests and the possibility of kernel fine-tuning.

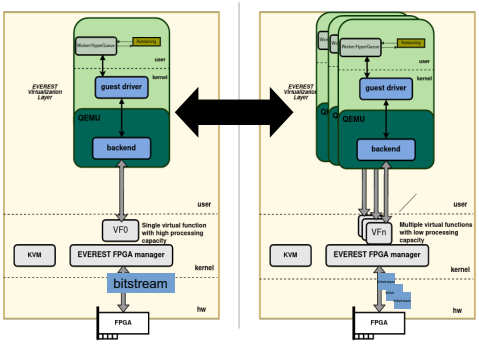

Fig. 6. Component diagram inside a physical node (virtualization perspective)

B. Virtualization infrastructure

We use a virtualization environment to allocate and manage resources across the heterogeneous EVEREST nodes. Figure 6 shows the components running on each physical node and the attached accelerators. It aims to offer the applications inside the Virtual Machine (VM/Guest) the same accelerated functions that would be physically accessed. We use `QEMU-KVM` as the hypervisor running in the host and `libvirtd` as the agent, which will expose the relevant `libvirt` API to the external components, e.g., the resource manager.

We use the SR-IOV (Single Root I/O virtualization) technique[2] to expose a Physical Function (PF) and several Virtual Functions (VFs). The PF provides the management interface to assign a VF to a separate VM. One VF can be assigned to a single VM, but many VFs can be assigned to the same VM. This approach results in near-native performance. The components inside the guest can utilize the VF like they were executing in a physical environment. One of the drawbacks of SR-IOV is that it is less flexible than other I/O virtualization methods, as it has a more static nature regarding the maximum number of defined VFs and the way resources are assigned to the VMs. To mitigate the latter downside, we design a mechanism that will receive a request from the EVEREST resource allocator and, depending on the exact situation, will perform dynamic plugging/unplugging of VFs to/from the VMs.

Both the autotuner, the runtime manager, and the resource allocator can interact with the virtualization infrastructure using `libvirt`. Thanks to the `libvirtd` daemon, the node where the hypervisor is installed can respond to queries about available resources and the system's current status. The autotuner can use this feature to make decisions.

C. Dynamic autotuning

The EVEREST autotuner, named mARGOt [8], is an application-level library that monitors the application performance during execution and selects the best configuration according to the execution environment (i.e., available hardware resources, resource usage, and data to be processed). The EVEREST mARGOt works by using knobs and metrics. The former are variables controllable by the library and are used to set the chosen configuration (e.g., application parameters or code variants). The metrics are observable variables whose value reflects functional and extra-functional properties for the application's execution. The mARGOt library requires describing the hardware used for the execution, including all the aspects of the current execution environment that can influence the choice of the best configuration (e.g., the number of CPU cores, their frequency, their utilization, and the presence of FPGA accelerators for the algorithm variants [10]). The dynamic autotuning framework selects the best set of knob values (e.g., application parameters or variants) based on metric values and execution environment status.

[2]Single Root I/O Virtualization and Sharing Specification

VII. Anomaly Detection

The SDK allows developers to deploy anomaly detection at any point throughout their workflow with minimal effort. Anomaly detection can serve as *input sanitization* to protect the models or to detect other security events. Developers are provided with two nodes: model selection and detection. In *model selection*, AutoML techniques are used to automatically find the best model and its best hyperparameters on the provided data, using the Tree-structured Parzen Estimator algorithm for hyperparameter sampling of Optuna [1]. After a specified amount of time, the node will output the best-found model. The *detection* node receives the same data as the model selection node and runs the model on the provided data to detect anomalies. As output, the node produces a JSON file containing the indexes of data points that are considered anomalous, upon which further action can be taken. The model is continuously updated with current data. The library handles most common data formats, but a simple configuration file must be provided to load the data if a special format is used or some specific subset of data should be processed.

VIII. Prototypes and Technical Highlights

This section describes the EVEREST prototypes and how we plan to apply the SDK features. Finally, we discuss the technical highlights of the project.

Accelerated WRF. WRFDA currently assimilates data from radar and authoritative and non-authoritative weather stations. We also derived customized configurations to understand the impact of atmospheric variables on the related workflows. The WRF model runs on HPC resources through LEXIS with a historical dataset, including about two years of prediction. The hardware acceleration of the WRF model's radiative-convective processes will lead to better predictions.

Renewable-energy prediction. Currently, the weather model is connected with the prediction application with automatic transfers of the high volume of WRF data to the rest of the application. This complete application is used to fine-tune the algorithms, parameters, inputs, and WRF runs to improve accuracy in a backtesting scenario. The possibility of increasing the number of WRF runs with more updates and getting closer to power delivery is a crucial advantage.

Air-quality monitoring. The current deployment of the air-quality use case combines a WRF-based weather forecast and an air-quality forecast using the ADMS model[3]. EVEREST will manage FPGA execution for the WRF part to reduce the execution time and manage ensemble weather forecasts. An ensemble can be created by using i) different weather global forecasts as input, ii) different physical modules in the WRF configuration, or iii) perturbations in initial 3D weather fields.

Traffic modeling. For Map-Matching, we conducted an exploration using the EVEREST SDK to generate hardware-accelerated implementations of the individual sub-kernels and to transparently decide at compile time where to allocate the kernels (FPGA or CPU), increasing the flexibility of the application based on the available target nodes. We also implemented the PTDR kernel on a compute cluster with Alveo u55c FPGAs, integrating it with the overall simulator. We also tested this component with the virtualization layer.

EVEREST technical highlights. The EVEREST project demonstrated that big data applications can benefit from FPGA accelerators, but their design and optimization demand integrated tools. Due to the variety of input languages, the convergence obtained with the MLIR flow is essential to unify the design of the FPGA architectures and decouple optimizations at the compiler level (for the application's functionality) and the platform level (for efficient data management). Custom data formats can significantly speed up the computation, trading off resource requirements and accuracy. The virtualized runtime environment enables seamless support for nodes with different hardware characteristics. In conclusion, coordinated tools (like the ones in the EVEREST SDK) are essential and can significantly improve the designer's productivity.

IX. Concluding Remarks

This paper presented the main result of EVEREST, i.e., the EVEREST SDK, which is a framework for big data applications on FPGA-based clusters. We presented the three major features: the compilation framework, the runtime environment, and the anomaly detection service. We also discussed how the SDK can be applied to the project's use cases and the technical highlights coming from the project.

Acknowledgements

This project has received funding from the EU Horizon 2020 Programme under grant agreement No 957269 (EVEREST).

References

[1] T. Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proc. of ACM SIGKDD*. 2019.

[2] P. Bauer et al. "The quiet revolution of numerical weather prediction". In: *Nature* 525.7567 (2015).

[3] T. Chen et al. "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning". In: *Proc. of OSDI*. 2018.

[4] S. Ertel et al. "Compiling for Concise Code and Efficient I/O". In: *Proc. of CC*. 2018, pp. 104–115.

[5] S. Ertel et al. "STCLang: State Thread Composition as a Foundation for Monadic Dataflow Parallelism". In: *Proc. of Haskell*. 2019.

[6] F. Ferrandi et al. "Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications". In: *Proc. of DAC*. 2021, pp. 1327–1330.

[7] K. F. A. Friebel et al. "BASE2: An IR for Binary Numeral Types". In: *Proc. of HEART*. 2023.

[8] D. Gadioli et al. "mARGOt: A Dynamic Autotuning Framework for Self-Aware Approximate Computing". In: *IEEE Transactions on Computers* 68.5 (2019).

[9] S. Karol et al. "A Domain-Specific Language and Editor for Parallel Particle Methods". In: *ACM Trans. on Mathematical Software* 44.3 (2018).

[10] N. Khouzami et al. "Model-based Autotuning of Discretization Methods in Numerical Simulations of Partial Differential Equations". In: *Journal of Computational Science* (2021).

[11] C. Lattner et al. "MLIR: Scaling compiler infrastructure for domain specific computation". In: *Proc. of CGO*. 2021.

[12] R. Murillo et al. "Generating Posit-Based Accelerators With High-Level Synthesis". In: *IEEE TCAS-I* 70.10 (2023).

[13] R. Nane et al. "A Survey and Evaluation of FPGA High-Level Synthesis Tools". In: *IEEE TCAD* 35.10 (2016).

[14] J. G. Papastavridis. *Tensor calculus and analytical dynamics*. Routledge, 2018.

[15] C. Pilato et al. "EVEREST: A design environment for extreme-scale big data analytics on heterogeneous platforms". In: *Proc. of DATE*. 2021, pp. 1320–1325.

[16] C. Pilato et al. "System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip". In: *TCAD* 36.3 (2017).

[17] J. G. Powers et al. "The weather research and forecasting model: Overview, system efforts, and future directions". In: *Bulletin of the American Meteorological Society* 98.8 (2017).

[18] B. Ringlein et al. "Advancing Compilation of DNNs for FPGAs Using Operation Set Architectures". In: *IEEE Computer Architecture Letters* 22.1 (2023), pp. 9–12.

[19] B. Ringlein et al. "DOSA: Organic Compilation for Neural Network Inference on Distributed FPGAs". In: *Proc. of EDGE*. 2023.

[20] B. Ringlein et al. "System Architecture for Network-Attached FPGAs in the Cloud using Partial Reconfiguration". In: *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. 2019, pp. 293–300.

[21] B. Ringlein et al. "ZRLMPI: A Unified Programming Model for Reconfigurable Heterogeneous Computing Clusters". In: *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2020, pp. 220–220.

[22] N. A. Rink et al. "CFDlang: High-level Code Generation for High-order Methods in Fluid Dynamics". In: *Proc. of RWDSL*. 2018, pp. 1–10.

[23] N. A. Rink et al. "TeIL: a type-safe imperative Tensor Intermediate Language". In: *Proc. of ARRAY*. 2019.

[24] S. Soldavini et al. "Automatic creation of high-bandwidth memory architectures from domain-specific languages: The case of computational fluid dynamics". In: *ACM TRETS* 16.2 (2023).

[25] S. Soldavini et al. "Iris: Automatic Generation of Efficient Data Layouts for High Bandwidth Utilization". In: *Proc. of ASPDAC*. 2023.

[26] S. Soldavini et al. *Platform-Aware FPGA System Architecture Generation based on MLIR*. 2023. arXiv: 2309.12917.

[27] F. Suchert et al. "ConDRust: Scalable Deterministic Concurrency from Verifiable Rust Programs". In: *Proc. of ECOOP*. 2023.

[3] http://cerc.co.uk/environmental-software/ADMS-model.html