LEGaTO: First Steps Towards Energy-Efficient Toolset for Heterogeneous Computing

Adrian Cristal, Osman S. Unsal, Xavier Martorell, Paul Carpenter, Raul De La Cruz, Leonardo Bautista, Daniel Jimenez, Carlos Alvarez, Behzad Salami, Sergi Madonar (BSC), Miquel Pericàs, Pedro Trancoso (Chalmers), Micha vor dem Berge, Gunnar Billung-Meyer, Stefan Krupop, Wolfgang Christmann (CHR), Frank Klawonn, Amani Mihklafi (HZI), Tobias Becker, Georgi Gaydadjiev (Maxeler), Hans Salomonsson, Devdatt Dubhashi (MIS), Oron Port, Elad Hadar, Yoav Etsion (Technion), Christof Fetzer (TUD), Jens Hagemeyer, Thorsten Jungeblut, Nils Kucza, Martin Kaiser, Mario Porrmann (UBI), Marcelo Pasin, Valerio Schiavoni, Isabelly Rocha, Christian Göttel, Pascal Felber (UniNE)

ABSTRACT

LEGaTO is a three-year EU H2020 project which started in December 2017. The LEGaTO project will leverage task-based programming models to provide a software ecosystem for Made-in-Europe heterogeneous hardware composed of CPUs, GPUs, FPGAs and dataflow engines. The aim is to attain one order of magnitude energy savings from the edge to the converged cloud/HPC.

1 INTRODUCTION

In the last couple of decades, technological advances in the ICT sector have been the dominant factors in global economic growth, not to mention an increase in the quality of life for billions of people. At the heart of this advance lies Moore's Law, which states that the number of transistors in an integrated chip will double every 18 to 24 months with each step in the silicon manufacturing technology node. However, due to fundamental limitations of scaling at the atomic scale, coupled with heat density problems of packing an ever-increasing number of transistors in a unit area, Moore's Law has slowed down in the last two years or so and will soon stop altogether [1]. The implication is that, in the future, the number of transistors that could be incorporated in a processor chip will not increase. This development threatens the future of the ICT sector as a whole. As a solution to this challenge, there recently have been dramatically increased efforts toward heterogeneous computing, including integration of heterogeneous cores on die (ARM), utilizing general-purpose GPUs (NVIDIA), combining CPUs and GPUs on the same die (Intel, AMD, ARM), leveraging FPGAs (Altera, Xilinx), integrating CPUs with FPGAs (Xilinx), and coupling FPGAs and CPUs in the same package (IBM–Altera, Intel–Altera). Heterogeneity aims to solve the problems associated with the end of Moore's Law by incorporating more specialized compute units in the system hardware and by utilizing the most efficient compute unit for each computation. However, while software-stack support for heterogeneity is relatively well developed for performance, it is severely lacking for power- and energy-efficient computing. Given that the ICT sector is responsible for ~5% of global electricity consumption [2], software *ACM Microsoft Word template* stack support for energy-efficient heterogeneous computing is critical to the future growth of the ICT industry. The primary ambition of the LEGaTO project is to address this challenge by starting with a Made-in-Europe mature software stack and by optimizing this stack to support energy-efficient computing on a commercial cutting-edge European-developed CPU–GPU–FPGA heterogeneous hardware substrate [3] and FPGA-based Dataflow Engines (DFE), which will lead to an order of magnitude increase in energy efficiency. The LEGaTO project will utilize a completely integrated software system stack supporting generalized tasks for low-energy, secure and reliable parallel computing. We foresee that optimization opportunities for low-energy computing can be maximized through the task abstraction.

The mature software stack that will be the baseline for development of the project is a task-based programming model family with a dataflow-based runtime. These task-based programming models, OmpSs [4] and XiTAO [5], are precursors and testing grounds for future versions of the popular OpenMP programming model.

Although the task-based programming model is by itself good for energy-efficient computing on heterogeneous substrates, we aim to further enrich the programming model and runtime for explicit support for energy-efficiency. The main idea is to attach resource requirements to parts of the computation and to execute them on dynamically constructed hardware places consisting of collections of cores and memories matching the resource annotations. Each piece of the computation is a generalized task that manages its own

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

control flow via an embedded scheduler. Resource requirements describe the needs of the application, such as number of cores, power, reliability, and security. The tasks are annotated with the resource requirements and with their input and output structures. These annotations are propagated through the system stack for seamless integration of the software with heterogeneous hardware consisting of CPUs, FPGAs, DFEs and/or GPUs, to identify the energy-optimal execution of the task at runtime. In order to achieve this goal, the project will develop tools to determine resources based on metrics such as FLOPS/Byte, reuse distance, power consumption, etc. for individual tasks. Furthermore, the project will develop support in the programming model and runtime for heterogeneity. This will be achieved by adding topology information at the task level allowing us to select appropriate accelerators and also compute nodes in scale-out environments. A task-based programming model with a dataflow runtime is a good match for low-power hardware since tasking seamlessly enables the dispatching of processing operations close to data, while the dataflow runtime execution model is well adapted for streaming accelerators such as FPGAs or DFEs. For DFEs, the programming model currently explicitly defines where DFE execution takes place. Adding dynamic runtime support will be compelling for reasons of productivity and energy efficiency.

Finally, the LEGaTO project will apply this energy-efficient software toolset for heterogeneous hardware to three use cases. The first use case will be healthcare. The project will not only demonstrate a decrease in energy consumption in the healthcare sector; it will also show that the toolset will increase healthcare application resilience and security; both of which are critical requirements in this area. As a second use case, the project will demonstrate ease of programming and energy savings possible through the use of the LEGaTO project software–hardware framework for IoT, smart homes, and smart cities applications. Sensitive sensor information and actuator instructions can be received and sent via the developed secure IoT gateway. A third use case will be based on machine learning (ML), where the project will demonstrate how to improve energy efficiency by employing accelerators and tuning the accuracy of computations at runtime. This use case will explore object detection using Convolutional Neural Networks (CNNs) for automated driving systems and CNN- and Long Short-Term Memory (LSTM)–based methods for realistic rendering of graphics for gaming and multi-camera systems. In addition, the machine learning use case will be used to further optimize the energy efficiency in the two other use cases, as well as within the runtime.

It is important to balance the advantage of a low-energy heterogeneous CPU/FPGA/GPU hardware platform with security and resilience challenges. We are therefore working on ensuring the resilience of the software stack running on this hardware, while simultaneously optimizing for performance and low power. For fault tolerance we would like to exploit the unique characteristics of the heterogeneous CPU/GPU/FPGA platform in the runtime; for example by replicating tasks intelligently on diverse processing elements exploiting the spatial/temporal slack; additionally, we will investigate energy-efficient selective replication where only the most reliability-critical tasks will be replicated. Furthermore, we will leverage the task programming model for detecting error propagation across task boundaries and walking the task dependency graph at runtime, which will help with failure root cause analysis. Finally, we will use the properties of the task model to design application-level energy-efficient checkpointing where only the necessary and sufficient data (declared at the task entry) will be checkpointed. For security, we will develop energy-efficient security-by-design by leveraging instruction-level hardware support for security (SGX in x86 and TrustZone in ARM) to accelerate software-based security implementations.

In the rest of the paper, we will detail the LEGaTO hardware platform which is composed of CPUs (both X86 and big.LITTLE heterogeneous ARM processors), GPUs and FPGAs. Then, we will do a deep-dive on LEGaTO programming models and on how they can increase programmer efficiency and application performance on FPGAs, first we will report first results for OmpSs@FPGA, and discuss the advantages of the DFiant Hardware Description Language for FPGAs. Then we will follow up with first directions in the LEGaTO runtimes, in particular the heterogeneity-aware OmpSs and XiTAO schedulers with emphasis on energy for the case of OmpSs and performance for the case of XiTAO. Finally, we will wrap up with two use cases, the LEGaTO smart home use case will discuss how taskification could be leveraged, and the healthcare use case will report the initial results on FPGA-based acceleration.

2 HARDWARE PLATFORM

For integration and evaluation of the tool-sets that are developed within LEGaTO, we will use the heterogeneous hyperscale server platform RECS|Box 4.0 [6]. The hardware server platform seamlessly integrates CPUs, GPUs, and FPGAs combined with a highly flexible communication infrastructure. Its modularity allows the RECS|Box to be scaled demand-oriented and thus adapt to changing requirements. Applications running on the platform can be optimized to distribute tasks to the most suited computing modules and to make use of dedicated accelerators. Cyber-physical systems and IoT will make a lot of data from their environment and their users available for new services and applications. This information needs to be securely stored and processed – partly locally and partly in the cloud. Utilizing standardized computer-on-module (CoM) form factors enables us to deploy the same hardware platforms that are used in the data centers also for edge computing. The RECS|Box integrates microservers based on x86, 32-bit and 64-bit ARM mobile/embedded processors and 64-bit ARM server processors using COM Express, Toradex Apalis and NVIDIA Jetson TX1/TX2 CoM formats. FPGA accelerators can be integrated as dedicated microservers, e.g., high-performance COM express boards with Intel Stratix10 SoCs are available as well as low-power Toradex Apalis boards with Xilinx Zynq. Additionally, PCIe-based accelerators with FPGAs or GPUs can be integrated into the platform as well as PCIe attached storage. A unique feature of the modular architecture is the integrated high-speed communication infrastructure, which is based on a dedicated high-

speed low-latency communication network. It connects to the CPU-/GPU-based microservers via PCIe and to the FPGA-based microservers via their high-speed serial interfaces. Depending on the involved communication partners, it utilizes either integrated PCIe-based packet switches for host-to-host communication or asynchronous crosspoint switches, which allow connections between microservers independent of the used protocol. In addition to direct communication between the different microservers it also supports connection to storage or I/O-extensions, allowing easy integration of PCIe-based extension cards like GPGPUs or storage subsystems. Using these communication facilities, accelerators can be flexibly attached to different compute nodes and can be combined into larger, virtual units. At run-time, the communication topology can be reconfigured and adapted to changing application requirements via the middleware. Using OpenStack as a middleware layer allows for accessing the hardware and deploying the applications in a uniform way.

All hardware components equipped which rich sensorization, providing fast and easy access to power, voltage, and temperature on device level, microserver level, as well as server level. Dedicated microcontrollers, integrated into the server platform, are used for data aggregation and data preprocessing. Combined with the possibility to monitor selected components with a sampling rate of up to 1 MSPS, the platform is an ideal reference for evaluation of performance and efficiency and to characterize individual tasks at run-time. Within LEGaTO, especially cost-efficient realizations for edge computing are targeted, which can be easily realized due to the high modularity of the RECS|Box.

The second testbed in LEGaTO is based on Maxeler Dataflow Engines (DFEs). With its DFEs, Maxeler pioneers a multiscale dataflow computing paradigm, which fundamentally differs from the classical Von Neumann control-flow oriented paradigm where instructions are processed sequentially on a general purpose CPU. On a DFE, data is streamed from memory and passes through a pipeline of arithmetic and logic operators without any kind of control mechanisms. Arithmetic operations are simply carried out as data passes over operators in the pipeline. The entire dataflow structure can be deeply pipelined while maintaining an overall throughput of one result per clock cycle. Thousands of operations can be performed in parallel, dedicating all of the available chip area to computation. DFEs are extremely efficient for large-scale computations with a static compute graph. They have shown to deliver one to two orders of magnitude improvement in performance and energy efficiency over conventional CPU servers in a range of application domains.

The current generation MAX5 DFEs are based on a large Xilinx VU9P FPGA that provides the reconfigurable substrate for creating the dataflow pipeline. The DFE also contains 48GB DDR4 DRAM and a 100 GbE networking port. DFEs are PCIe cards that can be integrated into a range of different products. In Maxeler MPC-C series systems, up to 4 DFEs are integrated into a conventional dual socket CPU server, combining DFEs with server-grade Intel or AMD CPUs. A far more flexible system architecture is provided by Maxeler MPC-X series systems, where eight DFEs are incorporated into a dense 1U chassis, forming a pure dataflow appliance. This system is connected to conventional CPU servers via an Infiniband network. Inside the MPC-X, the DFEs are also directly connected through MaxRing.

A Maxeler MPC-X dataflow system has been delivered to the Jülich Supercomputing Centre and is available for academic users. Maxeler MAX5 DFEs are also compatible with Amazon EC2 F1 instances and applications developed for MAX5 can be migrated seamlessly to the Amazon Cloud.

3 LEGaTO PROGRAMMING MODELS AND PROGRAMMER EFFICIENCY

3.1 OmpSs@FPGA

The OmpSs Programming Model supports the execution of heterogeneous tasks written in OpenCL, CUDA (for GPUs), and C/C++ (for FPGAs). Both OpenCL and CUDA options require the programmer to provide the OpenCL or CUDA code. In the case of the FPGAs, the programmer provides C/C++ code to be transformed into Verilog/VHDL by the FPGA vendor High-Level Synthesis tools (HLS). In the case of the Xilinx platforms, Vivado HLS accomplishes this transformation. Programmers can also provide the particular pre-generated bitstream to be executed in the FPGA, just by ensuring that the interface of the bitstream functionality is compatible with the OmpSs@FPGA interface.

FPGA exploitation with OmpSs consists of annotating the source code with OpenMP-like directives.

Figure 1 shows the code of matrix multiplication annotated with the OmpSs directives (target and task), in order to use the matrix_multiply function as a task to be transformed as an FPGA IP accelerator. In the matrix multiplication code, Vivado HLS directives provide hints to the Vivado translator to improve the VHDL generation, and obtain a high-performance bitstream.

```
#define BS 128

#pragma omp target device(fpga) copy_deps onto(0) num_instances(3)
#pragma omp task in(a,b) inout(c)
void matrix_multiply(float a[BS][BS], float b[BS][BS],float c[BS][BS])
{
#pragma HLS inline
  int const FACTOR = BS/2;
#pragma HLS array_partition variable=a block factor=FACTOR dim=2
#pragma HLS array_partition variable=b block factor=FACTOR dim=1
  // matrix multiplication of a A*B matrix
  for (int ia = 0; ia < BS; ++ia)
    for (int ib = 0; ib < BS; ++ib) {
#pragma HLS PIPELINE II=1
      float sum = 0;
      for (int id = 0; id < BS; ++id)
        sum += a[ia][id] * b[id][ib];
      c[ia][ib] += sum;
    }
}

…
for (i_b=0; i_b<NB_I; i_b++)
  for (j_b=0; j_b<NB_J; j_b++)
    for (k_b=0; k_b<NB_K; k_b++)
      matrix_multiply(AA[i_b][k_b], BB[k_b][j_b], CC[i_b][j_b]);
…
```

Figure 1. Matrix multiply with OmpSs and Vivado HLS annotations

3.1.1 OmpSs@FPGA Experiments

With this benchmark, several configurations of IP cores were tested on the FPGA using different clock speeds. The following table summarizes the experiments conducted:

IPs configuration	1*256, 3*128	Number of instances * size
Frequency (MHz)	200, 250, 300	Working frequency of the FPGA
Number of SMP cores	SMP: 1 to 4 FPGA: 3+1 helper, 2+2 helpers	Combination of SMP and helper threads
Number of FPGA helper threads	SMP: 0; FPGA: 1, 2	Helper threads are used to manage tasks on the FPGA
Number of pending tasks	4, 8, 16 and 32	Number of tasks sent to the IP cores before waiting for their finalization

The experimentation environment is the AXIOM board, consisting of the Xilinx Zynq Ultrascale+ chip, with 4 ARM Cortex-A53 cores, and the ZU9EG FPGA.

Figure 2 shows the evaluation in GFlops of the different alternatives of matrix multiplication configurations for 2048x2048 matrices using different block sizes:128x128 and 256x256. On the SMP cores, matrix multiplication used the OpenBLAS SGEMM kernel to multiply the matrices in parallel in the same blocked fashion as the application did on the FPGA. A single ARM Cortex A53 core delivered roughly 3 GFlops (Figure 2 "No IP" bars 1 core). Similarly, four ARM cores achieved 11.7 GFlops, showing good scalability of the OmpSs infrastructure on the SMP environment.

When adding an IP core (performance-oriented hardware design) of block size 256x256 running at 200 MHz, the performance was boosted to 25.8 GFlops. The 200 MHz FPGA implementation of the 256x256 block added up to 14 GFlops. In this latter case, one helper thread was used to execute FPGA tasks but also SMP tasks if there were not enough FPGA tasks for execution or the FPGA was busy, and 3 worker threads were running SMP tasks. This was due to the `implements` clause shown above, thus allowing a heterogeneous parallel execution of tuned tasks for both SMP and FPGAs.

Starting from this point, we increased the accelerator frequency to 250 and 300 MHz, which showed also an additional boost in performance. Additionally, we allowed the runtime system to provide up to 16 tasks to the FPGA before waiting for the previous tasks to be finished. Tuning this value also provided a further increase on performance. Specifically, for block sizes of 256x256 elements running at 300 MHz, the performance increased from 32.9 to 35.7 GFlops. This additional improvement was obtained from the reduction in the number of synchronizations needed by the runtime. Reducing the amount of synchronizations contributed to reduce the overhead of task management.

Figure 2 also shows the performance of 3 IP cores running in parallel on the FPGA. The additional parallelism yielded better performance. When running at 200 MHz, and with up to 16 pending tasks, the performance was equivalent to the one using a block size of 256x256 elements and 8 pending tasks. When moving to 300 MHz, the runtime behavior showed that it is important to have at least 2 helper threads available. Otherwise, the performance would have not been improved at all, since having only one helper thread did not provide enough FPGA tasks to the accelerators. In any case, those two helper threads are implemented in a hybrid way. That is, they can decide to run SMP tasks if the accelerators are busy, in addition to the two worker threads that are also executing SMP tasks. This combination of helper and worker threads achieved the best performance of the MxM. Observe that the combinations of 4x2, 16x2, and 32x2 (pending tasks and helper threads) keep increasing in performance, while the alternatives of 4x1, 16x1 and 32x1, which used a single helper thread, did not scale anymore.

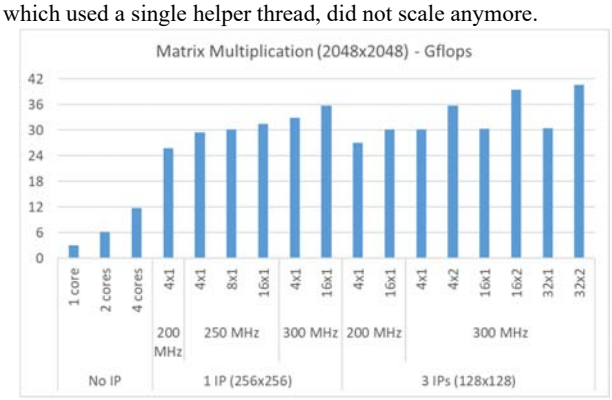

Figure 2. Evaluation of the matrix multiplication benchmark on the AXIOM board

3.2. DFIANT HDL

DFiant [7] is a hardware description language (HDL) and one of the LEGaTO programming models. For a long time, the dominating HDLs have been Verilog, System Verilog, and VHDL, and they all provide the same register transfer-level (RTL) hardware design abstraction. An RTL language burdens designers with explicitly clocked constructs that do not distinguish between design functionality and implementation constraints (e.g., timing, target device). For example, VHDL and Verilog constructs require designers to explicitly place a register, regardless if it is part of the core function- ality (e.g., a state-machine state register), an artifact of the timing constraints (e.g, a pipeline register), or an artifact of the target in- terface (e.g., a synchronous protocol cycle delay). These semantics narrow design correctness to specific timing restrictions, while vendor library component instances couple the design to a given target device. Evidently, formulating complex portable designs is difficult, if not impossible. Finally, these older languages do not support modern programming features that enhance productivity and correctness such as polymorphism and type safety. High-level synthesis (HLS) tools such as Vivado HLS [8], and high-level HDLs such as Bluespec SystemVerilog [9] and Chisel [10] attempt to bridge the programmability gap. While these tools

and languages tend to incorporate modern programming features, they still mix functionality with timing and device constraints, or lack hardware construction and timed synchronization control. On one hand, Chisel and Bluespec constructs explicitly pipeline designs. And on the other hand, Vivado HLS C++ constructs cannot directly support a simple task as toggling a led at a given rate. Such tools and languages, therefore, fail to deliver a clean separation between functionality and implementation that can yield portable code, while providing general purpose HDL constructs.

Figure 3 summarizes the primary programmability pros and cons of RTL and HLS languages with their targeted programming domains, architectures and accelerators, respectively. DFiant aims to bridge over the programmability gaps by combining constructs and semantics from software, hardware and dataflow languages. DFiant is not an RTL language, nor is it a sequential HLS language such as C. Instead, the DFiant programming model accommodates a middle-ground approach between low-level hardware description and high-level sequential programming.

DFiant is a modern HDL whose goal is to improve hardware programmability and designer productivity by enabling designers to express truly portable and composable hardware designs. DFiant decouples functionality from timing constraints (in an effort to end the "tyranny of the clock" [11]). DFiant offers a clean model for hardware construction based on its core characteristics: (i) a clock-agnostic dataflow model that enables implicit parallel data and computation scheduling; and (ii) functional register/state constructs accompanied by an automatic pipelining process, which eliminate all explicit register placements along with their direct clock dependency.

DFiant is implemented as a Scala library and relies on Scala's strong, extensible, and polymorphic type system to provide its own hardware-focused type system (e.g., bit-accurate dataflow types, input/output port types). The library performs two main tasks: the frontend compilation, which translates dataflow variable interactions into a dependency graph; and the backend compilation, which translates the graph into a pipelined RTL code and a TCL constraints file, followed by a hardware synthesis process using commercial tools. DFiant can be used in any Scala-compatible IDE, including the LEGaTO eclipse-based tool-chain.

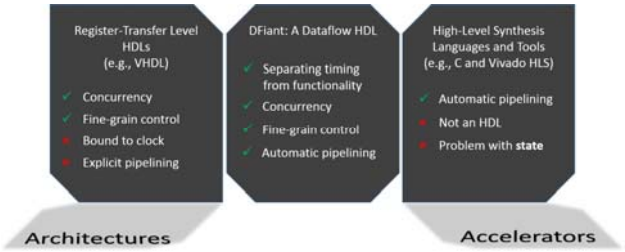

Figure 3. DFiant fills in the programmability gaps

3.3 Maxeler MaxCompiler

Developing applications for Maxeler DFEs requires identifying the computationally challenging part of the application and porting this part to a dataflow model. This model is developed in a Java-based meta-language called MaxJ. MaxJ programming adopts Java syntax but is in principle different from regular Java programming or other imperative programming paradigms. Compiling MaxJ code does not produce a Java application; instead, it leads to the generation of dataflow kernels. Both the compute kernels handling the data-intensive part of the application and the associated manager, which orchestrates data movement between kernels and external interfaces, are written using MaxJ. Developing a dataflow application therefore involves implementing three parts:

1) A CPU host application typically written in C/C++, Matlab, Fortran, Python, R, etc;
2) A number of dataflow kernels written in MaxJ;
3) A manager described in MaxJ.

Maxeler MaxCompiler is a comprehensive development, debug and simulation environment for developing dataflow applications in MaxJ. In order to provide seamless DFE integration with host applications, Maxeler provides a Simple Live CPU (SLiC) API, and MaxelerOS, a runtime layer between the SLiC API, the Linux operating system and the DFE hardware. MaxIDE2.0 is an IDE with dedicated features for developing, debugging, visualizing, and optimizing dataflow applications. Applications developed in MaxJ are independent of the underlying FPGA technology and are always forward compatible to newer DFE generations. As of version 2018.1, MaxCompiler also supports building applications for Amazon EC2 F1 instances. MaxJ dataflow applications can be moved from a conventional Maxeler DFE to F1 by simply changing the build target.

4 LEGaTO RUNTIMES AND ENERGY-PERFORMANCE-EFFICIENCY

4.1. OmpSs Heterogeneous Task Scheduler

Heterogeneous computing facilities such as CPUs, GPUs and FPGAs offer various trade-offs in terms of performance, adequacy to the task, potential for parallelism, power consumption, and even dependability and security properties. These trade-offs can vary depending on the application and the specific workload being processed. The LEGaTO Project develops a low-energy toolset for heterogeneous computing that includes a smart scheduler for tasks across different resources. To enable such smart scheduling of tasks, we need several components which we are developing in the context of the LEGaTO project.

First, we need an appropriate programming model with fine-granular computations, so that we can quickly launch and move tasks across multiple heterogeneous nodes. It must additionally provide support for parallel computation, as well as for cross-cutting properties such as fault-tolerance and security. To that purpose, we will base our programming model on OmpSs (https://pm.bsc.es/ompss) and extend it with the missing features necessary for LEGaTO.

Second, we need an energy model able to derive and predict the power consumption of tasks operating with some given input data on specific hardware resources.

Third, we need a monitoring framework to measure and keep track of the power consumption of the tasks running in the system.

The information produced by this framework will be used to refine the energy model and drive the scheduling policies based on actual power usage.

Finally, and most importantly, we need an actual scheduling framework to orchestrate the placement and execution of tasks on the hardware resources.
The scheduling strategies will use information from the monitoring framework evaluating the actual power usage, as well as from the energy model to help with prediction and take informed placement decisions.
Early results with preliminary version of the monitoring framework reveal important differences in performance and energy consumption, offering interesting trade-offs for task scheduling. As an example, Figure 4 shows the performance results of executing simple workloads across different hardware architectures. This data tends to indicate that, by placing tasks on energy-efficient processor, one can obtain non-negligible energy savings with only limited performance penalty.

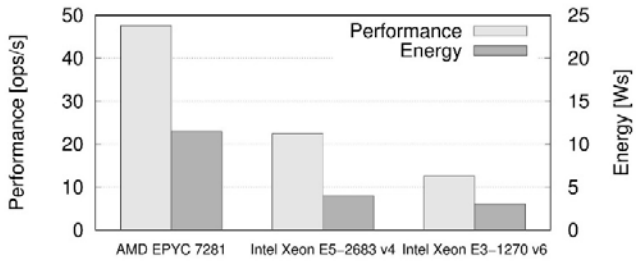

Figure 4. Performance and energy used by the same program on three different machines

Regarding the scheduling framework, we leverage experience and knowledge gained from previous work on energy-aware scheduling. We are currently extending our existing scheduler policies to render the framework heterogeneity- and energy-aware. In order to achieve this goal we rely on multiple types of power meters (hardware and software ones) for monitoring the running system and to build up and continuously update a knowledge base to be consulted by the scheduler.

4.2. XiTAO Heterogeneous Task Scheduler

The Chalmers' High Performance Computer Architecture group will contribute to the LEGaTO toolchain in three directions: 1) development of energy-efficient scheduling techniques, 2) development of portable abstractions for data locality, and 3) runtime performance monitoring and feedback.
This work will extend the work developed in the context of the XiTAO research runtime [12]. XiTAO is a task-based runtime that generalizes the concept of task into a parallel computation with arbitrary (*elastic*) resources. This type of generalized task is called a TAO (for Task Assembly Object). By matching task requirements with hardware resources (cores, memory, etc) at runtime, XiTAO targets high parallelism and provides constructive sharing and interference freedom. Overall, this strategy improves the energy efficiency of the computation. In addition to these, XiTAO provides a data locality abstraction called *software topology*. This abstraction allows the programmer to overlay a virtual ("software") topology, such as a 1D line, 2D plane or 3D cube, on top of the task graph, and assigns each task a location (an "address") in this software topology. The XiTAO runtime uses the address information to dynamically select locality-aware schedules in a way that is independent of the actual hardware topology, and thus support dynamically reconfigurable topologies. The main goal in the context of LEGaTO is to extend the XiTAO technologies to support resource heterogeneity.
During the early months of the project, one master thesis [13] developed at Chalmers has evaluated the potential of simple scheduling heuristics with and without knowledge of hardware heterogeneity. This work focused on the HiKey 960 board featuring the HiSilicon Kirin 960 big.LITTLE SoC. The XiTAO runtime was ported to this platform (thus adding support for ARM) and a set of microbenchmarks was developed to test if XiTAO's resource elasticity can be an effective method to address heterogeneity. The work considered both criticality-based and weight-based scheduling, in which tasks were assigned to big cores when they were deemed to be either in the critical path (criticality-based), or when their relative speed-up over the LITTLE cores was larger than a threshold (weight-based). To assess the capabilities of each core, the criticality-based scheme was implemented also in a topology-unaware way. In this case, the runtime constructs a *performance table* in which the execution time of each task is recorded depending on the identifier of the first core in the team and the amount of cores used.
The microbenchmarks consisted of randomly generated DAGs with varying amount of parallelism and were evaluated compared to a homogeneous random work stealing strategy. Particularly for cases of low average parallelism (=1.62), large average speed-ups ranging from 1.29x to 2.79x were observed depending on the resources statically allocated in the homogeneous scheduler. For medium parallelism (=3.03) the speed-ups were slightly lower, ranging from 1.27x to 2.03x. Finally, in the case of high parallelism (=8.06) we observed speed-ups ranging from 1.1x to 1.28x. This behavior is expected since the higher the parallelism, the lesser the impact of task criticality. The performance table proved to be useful to identify not only the core capabilities but also the current load of the system. The latter is achieved by updating the execution time each time a new task completed execution. This allows the scheme to learn about the average parallelism of the application that is being executed and also to understand if an interfering workload is currently running on some of the cores.
In addition to novel runtime techniques for efficient scheduling, Chalmers' will also develop compiler technologies to aid in the development of TAOs. These techniques will analyze static task graphs obtained from OmpSs and identify task subgraphs that can be encapsulated into TAOs. This will enhance productivity by following the LEGaTO philosophy of generating the LEGaTO binary fully from a single source file written in the OmpSs language.

4.3. BRAM undervoltage usage in FPGA

Usually, chip vendors are forced to add conservative voltage guardbands to ensure the worst-case process and environmental scenarios. Eliminating this voltage guardband by undervolting below the standard nominal level is an effective solution for improving power and energy efficiency in digital circuits.

However, aggressive undervolting without accompanying frequency scaling leads to timing related faults, potentially undermining the power savings. We investigated the aggressive voltage underscaling to explore the power and reliability trade-offs for a commercial heterogeneous architecture from Xilinx, a main vendor, i.e., Zynq platform. More specifically, our preliminarily concentration is on the on-chip memories or Block RAMs (BRAMs) of Programmable Logic (PL) part. As can be seen in Figure 5; on ZC702, we experimentally observed an extremely conservative voltage guardband of 39% of the nominal level. It means that undervolting until Vmin=0.61V, while Vnominal=1V, no fault occur, while BRAMs power consumption is reduced by an order of magnitude. However, further undervolting, faults manifest with an exponentially increasing rate up to 172 faults per 1Mbit and with fully non-uniformly distributed among different BRAMs.

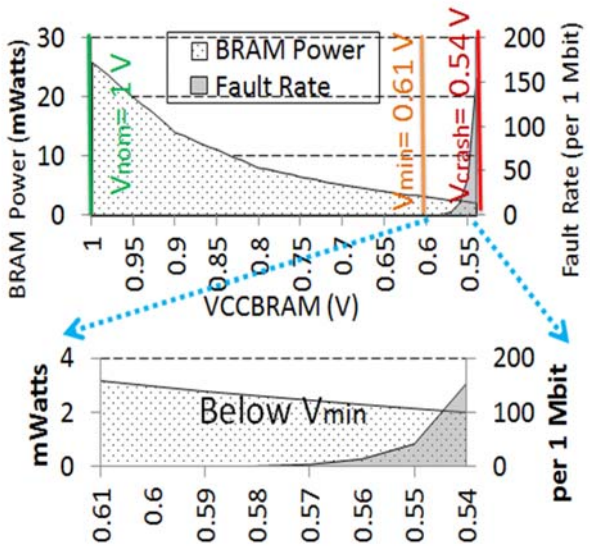

Figure 5. Power and Reliability trade-off by undervolting ZC702 BRAMs.

Motivated by this preliminary study on ZC702, we aim to take the advantage of aggressive voltage scaling on LeGaTO hardware platforms and apply it to our use cases, while tools of the project will handle the run-time voltage scaling assignments to achieve energy efficiency without compromising the performance and reliability issues.

5 LEGaTO USE CASES

5.1 Smart Home Use Case

Current smart living environments are based on the simple automation of subsystems consisting of sensors, information processing, and actuators. New approaches mainly driven by large enterprises push big-data based approaches, collecting as much information about the user as possible to derive the current action and to anticipate future behavior. The development of assisted living can be seen as a move from isolated applications realized as simple embedded systems, towards cyber-physical systems fusing and processing large amounts of data from a high number of distributed smart devices. Additionally, the smart home has to process interaction of different users simultaneously; conflicting actions have to be recognized (e.g., one user opening a window and another one closing it again) and compromises can be suggested. Different interaction schemes can be combined adaptively, e.g., switching from touch to speech interaction while cooking or using text-based output while phoning. Providing this functionality is highly computational intensive. Since the collected data contains personal and highly sensitive information, cloud-based processing is undesirable. To address these privacy issues, we target resource-efficient edge computing. The taskification of the application – coupled by the LEGaTO runtime, performing energy-aware load balancing and scheduling – leads to the most energy efficient execution on the heterogeneous hardware platform, combining CPUs, GPUs and FPGAs in an appropriate way.

For everyday use, smart home environments require an intuitive and comfortable interface for user interaction. Depending on the desired functionality of an individual assistance system, specific modalities for interaction are preferred by the user, e.g., touch, speech, gestures, or even emotions. Within LEGaTO, a uniform configurable platform for distributed interaction and information processing in smart homes is developed. The platform can be configured to integrate arbitrary combinations of basic functionalities like face, object, and speech recognition. As an example implementation, this platform is used to realize a central human-machine interface for smart homes integrated in a wardrobe mirror. In addition to information and control of the state of the smart home environment (heating, ventilation and air conditioning (HVAC), security, illumination, and many more), the smart mirror provides cognitive abilities targeting more sophisticated assistive functionalities including, e.g., virtual try-on of a dress, guidance to tie a necktie, or interpretation of user intention.

For the user interface, the smart mirror prototype builds upon the open source project “magic mirror” (https://magicmirror.builders/). In addition to the basic visualization provided by this environment, our prototype integrates compute-intensive methods for face, object and speech recognition. Face recognition is used to provide personalized content for individual users. It is realized using deep convolutional networks, implemented in three steps with TensorFlow. In the first step, a pretrained graph from “WIDERFace” [14] is used to find faces in the input video stream. Subsequently, a feature representation is computed for every detected face using “FaceNet” [15]. Based on the extracted features, a classifier is trained to determine the identity of the respective person using “Scikit-learn” [16].

Speech recognition provides an easy and comfortable possibility for direct interaction with the smart home environment. The smart mirror provides a local system for speech recognition using “DeepSpeech” [17]. In addition to direct processing of recorded audio, other smart home devices can send their audio streams to the smart mirror, using it as an edge platform for the required recognition tasks. Object detection is another key feature for smart environments. For the proposed implementation, “YOLO” is used,

which is capable of recognizing more than 9,000 object categories [18].

As a reference implementation, the software is implemented on the RECS|Box using x86 processors and GPUs. For the final platform we target resource-efficient combinations of embedded CPUs, embedded GPUs, and FPGAs. The LEGaTO tool flow will be used to find optimized solutions for the different hardware-software combinations. The developed concepts used for the smart mirror can be easily transferred to other use cases, including entertainment systems, sports coaching, and cooking assistant [19]. In addition to the improvement of the overall user experience by enhancing naturalness of interaction and seamless or nearly invisible integration of assistive functionality, the ability of our homes to detect unexpected deviations from normal operation, unwanted behavior, or dangerous situations becomes more and more important. Deviation recognition enables especially elderly or cognitive impaired people to live a self-determined life as long as possible in their own home.

5.2 Healthcare Use Case

In order to better understand how infectious diseases spread and which effects vaccination strategies have, intensive simulation studies are needed [20]. For the analysis of real data from patients or mouse experiments with a limited number of samples but a large number of observed variables, Bayesian approaches are often better suited [21]. Simulation studies are computationally expensive because they have to be repeated very often in order to obtain reliable results. Bayesian inference relies on Markov chain Monte Carlo (MCMC) techniques in order to estimate the posterior distribution representing the essential outcome of a Bayesian analysis. MCMC techniques are computationally very expensive. Therefore, simulation studies as well as Bayesian analysis highly benefit from concurrency and special hardware that supports the underlying computations. A main goal of this use case within the LEGaTO project is to achieve a speed-up of the computations to enable simulations and analyses that would not be feasible with ordinary implementations and hardware. First prototypes have demonstrated that a speed-up by a factor of more than 1000 is possible.

6 About LEGaTO

The H2020 LEGaTO (Low Energy Toolset for Heterogeneous Computing) project, grant agreement n° 780681, is funded by the European Commission with a budget of more than €5 million and will last three years from its beginning on 1 December 2017. The partners of the project are Barcelona Supercomputing Center (BSC, Spain), Universität Bielefeld (UNIBI, Germany), Université de Neuchâtel (UNINE, Switzerland), Chalmers Tekniska Högskola AB (CHALMERS, Sweden), Machine Intelligence Sweden AB (MIS, Sweden), Technische Universität Dresden (TUD, Germany), Christmann Informationstechnik + Medien GmbH & Co. KG (CHR, Germany), Helmholtz-Zentrum für Infektionsforschung GmbH (HZI, Germany), TECHNION Israel Institute of Technology (TECHNION, Israel), and Maxeler Technologies Limited (MAXELER, United Kingdom).

REFERENCES

[1] ITRS, International Technology Roadmap for Semiconductors 2.0: 2015 Edition, ITRS, 2015.

[2] W. Van Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet y P. Demeester, «Trends in Worldwide ICT Electricity Consumption from 2007 to 2012,» Comput. Commun., vol. 50, pp. 64-76, 9 2014.

[3] R. Griessl y e. al, «A Scalable Server Architecture for Next-Generation Heterogeneous Compute Clusters,» de 2014 12th IEEE International Conference on Embedded and Ubiquitous Computing, 2014.

[4] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell y J. Planas, «Ompss: a Proposal for Programming Heterogeneous Multi-Core Architectures.,» vol. 21, pp. 173-193, 6 2011.

[5] M. Pericàs, «ξ-TAO: A Cache-centric Execution Model and Runtime for Deep Parallel Multicore Topologies,» de Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, NY, USA, 2016.

[6] A. Oleksiak y e. al, «M2DC – Modular Microserver DataCentre with heterogeneous hardware,» Microprocessors and Microsystems, vol. 52, pp. 117-130, 2017.

[7] O. Port y Y. Etsion, «DFiant: A dataflow hardware description language,» de 2017 27th International Conference on Field Programmable Logic and Applications (FPL), 2017.

[8] Xilinx, Vivado High Level Synthesis User Guide, 2015.

[9] R. Nikhil, «Bluespec System Verilog: efficient, correct RTL from high level specifications,» de Formal Methods and Models for Co-Design, 2004. MEMOCODE '04. Proceedings. Second ACM and IEEE International Conference on, 2004.

[10] J. Bachrach y e. al, «Chisel: Constructing hardware in a Scala embedded language,» de DAC Design Automation Conference 2012, 2012.

[11] I. Sutherland, «The Tyranny of the Clock,» Commun. ACM, vol. 55, pp. 35-36, 10 2012.

[12] M. Pericàs, «Elastic Places: An Adaptive Resource Manager for Scalable and Portable Performance,» ACM Trans. Archit. Code Optim., vol. 15, pp. 19:1--19:26, 5 2018.

[13] H. Rohlin, «Performance-targeted Resource-aware TaskScheduling for Heterogeneous Platforms,» 2018.

[14] S. Yang, P. Luo, C. C. Loy y X. Tang, «WIDER FACE: A Face Detection Benchmark,» de IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[15] F. Schroff, D. Kalenichenko y J. Philbin, «FaceNet: A Unified Embedding for Face Recognition and Clustering,» CoRR, vol. abs/1503.03832, 2015.

[16] F. Pedregosa y e. al, «Scikit-learn: Machine Learning in Python,» CoRR, vol. abs/1201.0490, 2012.

[17] D. Amodei y e. al, «Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin,» de Proceedings of The 33rd International Conference on Machine Learning, New York, New York, USA, 2016.

[18] J. Redmon y A. Farhadi, «YOLO9000: Better, Faster, Stronger,» CoRR, vol. abs/1612.08242, 2016.

[19] A. Neumann y e. al, «"KogniChef": A Cognitive Cooking Assistant,» KI - Künstliche Intelligenz, vol. 31, pp. 273-281, 2017.

[20] A. Bakuli, F. Klawonn, A. Karch y R. Mikolajczyk, «Effects of pathogen dependency in a multi-pathogen infectious disease system including population level heterogeneity -- a simulation study,» Theoretical Biology and Medical Modelling, vol. 14, p. 26, 13 12 2017.

[21] F. Klawonn, T. Wüstefeld y L. Zender, «Statistical Modelling for Data from Experiments with Short Hairpin RNAs,» de Advances in Intelligent Data Analysis IX, Berlin, 2010.