This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2019.2952335, IEEE Design and Test

The AXIOM Project: IoT on Heterogeneous Embedded Platforms

Antonio Filgueras[2], Miquel Vidal[1,2], Marc Mateu[1,2], Daniel Jiménez-González[1,2], Carlos Álvarez [1,2], Xavier Martorell[1,2], Eduard Ayguadé[1,2], Dimitrios Theodoropoulos[4], Dionisios Pnevmatikatos[4,8], Paolo Gai[7], Stefano Garzarella[7], David Oro[5], Javier Hernando[1], Nicola Bettin[6], Alberto Pomella[6], Marco Procaccini[3] and Roberto Giorgi[3]

***Abstract*—The AXIOM project aims at providing an environment for Cyber-Physical Systems. Smart Video Surveillance targets public environments, involving real-time face detection in crowds. Smart Home Living targets home environments and access control. These applications are used as experimental use-cases for the AXIOM platform, currently based on the Xilinx Zynq-7000 SoCs. We have integrated the Xilinx Vivado HLS tool for the FPGA support within the OmpSs programming model, to enable OpenMP-like programming in the FPGA. This paper presents the programming environment, and the evaluation of the most computationally expensive parts of the target applications.**

***Index Terms*—IoT, OmpSs programming model, video surveillance, smart home, FPGA.**

I. Introduction

The AXIOM project aims to build a powerful node for IoT infrastructure. In the context of the project, we have selected two IoT application scenarios: Smart Video Surveillance and Smart Home Living. Those are being used to validate the Xilinx Zynq platforms as a good basis upon which we can build the services for IoT.

IoT nodes should have low-power characteristics, yet offer some performance level, to support local computations. In the Smart Video scenario, the node is able to detect the presence of faces in video streams, and send them to a server for additional checks, while in the Smart Home Living scenario, the node is able to detect a human voice in an audio stream and also to send it to a server for further actions to be taken. Usually, these tasks cannot be supported only by a small Symmetric Multi-Processor (SMP) chip, but we are experimenting with chips that also incorporate FPGAs, in particular those of the Xilinx Zynq-7000 family.

AXIOM IoT programming environment is based on the task-parallel OmpSs Programming Model. In the same way we use OmpSs [1] to leverage existing accelerated kernels in CUDA and OpenCL, we are developing the OmpSs@FPGA version. This flavor of OmpSs is able to generate IP cores for the FPGA fabric at compile time and execute them as if they were kernels running on an accelerator. With our OmpSs@FPGA infrastructure, kernels (function code annotated as FPGA tasks) are compiled automatically by using the FPGA tools available for High-Level Synthesis (HLS) for the target FPGA chip. Then, the OmpSs runtime automatically schedules work to be done to run on those IP cores. Data transfers to and from the IP cores are also automatically generated from the runtime system significantly simplifying the programmability.

This paper presents the following contributions:

- Description of the fully working OmpSs@FPGA ecosystem
- Porting of real scenarios to the full working environment
- Evaluation of the results of the project selected applications when executed using the full working environment and the FPGA resources compared against the traditional approach

In order to present this results we use OmpSs@FPGA that, to the best of our knowledge, is the first complete environment to achieve directive-based, OpenMP-like, parallel execution on FPGAs.

II. The AXIOM Hardware

The current AXIOM platform is a Zynq SoC of the Xilinx Zynq-7000 family. It features a dual-core ARM Cortex A9 processor, tightly coupled with the FPGA fabric. Some Zynq SoC boards have 4 transceivers, allowing to set small cluster systems through the low-cost but scalable high-speed interconnect infrastructure, originally presented in [2].

Computing resources in the AXIOM platform include the two ARM Cortex A9 cores, and the challenging FPGA fabric. We have defined a basic infrastructure for the FPGA fabric, containing the means to transfer data from the host SDRAM memory to the IP cores present in the FPGA. Figure 1 shows the block diagram and the connections among these elements.

As shown in Section III, the ARM Cortex A9 cores are used to run the Linux operating system, and the control side of the applications. The computation part of the applications is spawned as tasks on the IP cores programmed in the FPGA.

In this paper we evaluate the results of the two AXIOM project selected applications. As one of the goals of the project is to execute the applications in a sensor-attached board, the cluster capabilities of the system [3] are not used.

[1]Universitat Politècnica de Catalunya
[2]Barcelona Supercomputing Center
[3]University of Siena
[4]FORTH
[5]Herta Security
[6]VIMAR
[7]Evidence Srl
[8]Technical University of Crete

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2019.2952335, IEEE Design and Test

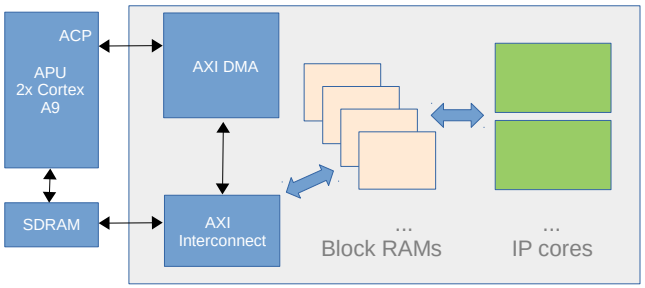

Fig. 1. The AXIOM compute node, including the APU cores, the system SDRAM, and the basic infrastructure for computing in the FPGA logic.

III. The AXIOM Software

During the development of the AXIOM project, we have defined the structure of our compilation toolchain (see Figure 2). The source code of the input OmpSs applications is analyzed by the Mercurium compiler [4], [5]. According to the OmpSs directives, the code is split into two parts. The source code controlling the execution goes to the host side (left). It is transformed to include calls to the Nanos++ runtime system to spawn the tasks that will invoke the IP cores in the FPGA. This code is compiled to binary using the GCC compiler for the ARM cores.

The code annotated as OmpSs tasks for the FPGA is also automatically converted to IP cores and to a bitstream (right side of Figure 2). After offloading the code into separate files, Mercurium automatically invokes the Xilinx Vivado HLS tool to generate HDL code, which is later synthesized with Xilinx Vivado.

The bitstream generated is used to configure the Zynq FPGA of the target board. The board runs Linux, and it incorporates the Xilinx FPGA driver. When executing the binary file, the Nanos++ runtime uses the specially designed DMA library to implement the data transfers to and from the FPGA, and to run the IP cores with the accelerated kernels. As the kernels are annotated as tasks, their execution also follows the correct ordering according to the task data-flow graph computed at runtime. The data necessary for the tasks (and the output data generated if it is the case) is also automatically transferred to/from the FPGA accelerators to the SMP cores simplifying this error-prone task.

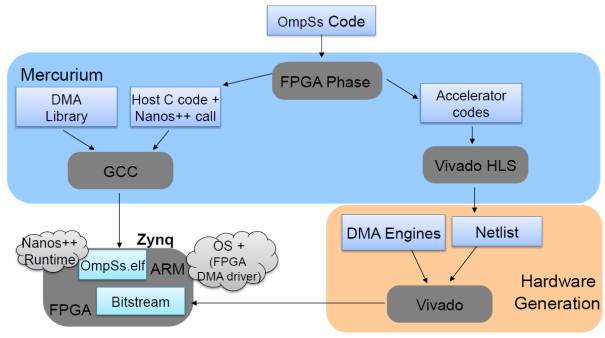

Fig. 2. The AXIOM toolchain.

IV. Programming the FPGAs

In the AXIOM project, the compiler toolchain runs on Intel x86_64 hosts. The reason is that Xilinx Vivado tools only run on this architecture. The set of tools needed to compile applications are:

- OmpSs Mercurium compiler, Intel version
- OmpSs Nanos++ runtime system, ARM version for linking
- Extrae instrumentation library, ARM version for linking
- The Xilinx board support package (BSP), for the target board
- Vivado HLS Design Suite (versions ranging from 2015.4 to 2017.1), from Xilinx
- ARM cross compiler, targeting Zynq-7000 boards, from the Xilinx SDK

Additionally, for the Zynq-7000 boards we provide (ARM versions to run on the board):

- OmpSs Mercurium compiler
- OmpSs Nanos++ runtime system
- Extrae instrumentation library
- libxdma library, supporting data transfers and execution of IPs on the FPGA
- Linux kernel xdma module, compatible with the latest Xilinx 4.6 Linux kernels

Programs written for the AXIOM environment spawn tasks on a set of functions, that the compilation flow will convert to IP cores to run on the FPGA. The Mercurium compiler does so while taking care of setting up the proper parameter interface between the A9 processors and the FPGA.

A tutorial sample application is shown in Figure 3.

```
#include <stdio.h>
#include <stdlib.h>

#pragma omp target device(fpga) copy_deps onto(0,1)
#pragma omp task depend(in:v_a[0;N], v_b[0;N]) \
                 depend(out:v_c[0;N])
void vector_multiply(float *v_a, float *v_b,
                     float *v_c)
{
   int i;
   for (i=0; i < N; i++)
       v_c[i] = v_a[i] * v_b[i];
}

int main(int argc, char *argv[])
{
   // Allocate and initialize vectors
   //    A and B, the source of the data
   //    C to receive the result of the vector product

   // Compute the vector product
   vector_multiply (A, B, C);
#pragma omp taskwait

   return 0;
}
```

Fig. 3. Vector multiply OmpSs code targeting the FPGA.

Source code is automatically transformed by the Mercurium compiler into two separate files. The code that runs in the Cortex A9 cores as an SMP application with calls to the Nanos++ runtime system, and the FPGA code to be compiled by the Xilinx Vivado HLS compiler onto the bitstream. Setting up this interface between the A9 cores and the FPGA is an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2019.2952335, IEEE Design and Test

error-prone task for a programmer. Instead, this is the type of transformation that a compiler can make automatically, based on the OmpSs directives. HLS pragma directives are parsed by the Mercurium compiler, and transparently passed through towards Vivado HLS compiler.

After that, the Nanos++ runtime will spawn a task to the FPGA whenever the annotated function is called. This task will follow the proper runtime order with respect to the remaining tasks (SMP or FPGA) thanks to the dependences annotated in the pragmas being executed in parallel whenever the runtime detect the possibility (due to the availability of both the tasks and the hardware resources).

V. Application Scenarios

The AXIOM platform hardware and software developed for the Cyber Physical Systems (CPS) paradigm is evaluated by means of two real-world use case scenarios: a Smart Video Surveillance (SVS) scenario, and a Smart Home Living (SHL) scenario.

Several resource intensive kernels of these scenarios have been parallelized using the OmpSs programming model.

A. Kernel from the SVS scenario

The SVS application was developed with the aim of demonstrating the validity of the AXIOM platform for implementing a state-of-the-art IoT edge computing architecture [6]. This application exploits advanced computer vision and video processing techniques for analyzing in real time all faces appearing on streams broadcast from CCTV surveillance cameras. By relying on the edge computing paradigm, it is possible to perform latency-oriented kernels, such as face detection and demographics estimation, close to the surveillance cameras directly on the low-power Xilinx Zynq SoC. On the other hand, resource intensive computations, such as face identification and template matching over databases populated with millions of subjects, are offloaded to a remote cloud data center, which is usually powered by hundreds of high-end discrete GPU and FPGA nodes.

Although it is currently possible to implement both face detection and demographic estimation using a combination of several CNNs, we found that it is still not possible to reasonably solve the problem of low-power real-time accurate face localization in HD images using pruned CNNs. Variations of region proposal networks, including Faster R-CNN, SSD or YOLO, still require devices featuring a large number of ALU or DSP units in order to beat traditional methods based on image descriptors tuned for the particular problem of face detection. As such, these methods are still uncompetitive if the whole accuracy/power/latency trade-off is considered [7].

Therefore, the SVS application relies on a cascade of boosted ensembles with LBP features [8] for locating faces and thus quickly discard image regions without faces. Local Binary Pattern (LBP) is a technique often used in Computer Vision, for image classification. Once faces have been located, they are used as an input to three different CNNs for performing the age, gender and ethnicity estimation. Convolutional Neural Network (CNN) is a feed-forward artificial neural network, used in Machine Learning environments. Both LBP cascade evaluation and the matrix multiply kernels (required for inferring the CNNs) were automatically parallelized using HLS and OmpSs pragma annotations.

B. Kernel from the SHL scenario

The application developed for the SHL scenario receives multimedia streams broadcast from devices, which are attached to the smart home local network. Then it decodes audio and video streams, and analyzes raw data using machine learning algorithms to gather information. The main tasks implemented on the SHL application are the identification of end users to grant or deny access to the home premises, and the interaction with the smart home system. User identification is performed by two biometric methods: speaker identification, which identifies users by analyzing acoustic features; and iris recognition, which recognizes users by analyzing the unique patterns found in the iris.

In order to allow a natural interaction between end-users and their homes, it is required that the identification phase does not interrupt the end-user actuation flow. The main objective is to avoid providing a slow feedback to end users. This challenge implies that the SHL application must deal with strict timing constraints to avoid unresponsiveness. In order to guarantee this requirement, the SHL application was profiled with the aim of substantially accelerating it.

The original code of the SHL application was developed in sequential C and C++.

Application profiling identified the acoustic *feature extractor module* and the *anisotropic smoothing module* as the most time consuming parts of the application.

The anisotropic smoothing operation is an image denoising technique that is aimed to preserve the edges of images while smoothing regions of uniform intensity. This type of filtering is usually used as a preprocessing stage of segmentation algorithms. Anisotropic smoothing is a very time consuming operation, required for the segmentation steps of iris recognition pipeline. To accelerate the execution of this module, the above mentioned algorithm was exhaustively analyzed to precisely determine the data flow and its corresponding dependences. Finally, it was annotated with OmpSs directives.

VI. Evaluation

A. Kernel from the SVS scenario

The cascade evaluation algorithm scans all the image performing the evaluation of LBP features with a 48×48 sliding window all over the x and y dimensions of the input image. This cascade of boosted ensembles is evaluated until the accumulated score exceeds the corresponding face detection threshold. In our baseline, the sequential implementation of the evaluation of LBP features is divided into two feature loop partitions for performance reasons. The first one (94 stages) finishes its processing as soon as the threshold is not met (`break` statement), and the second evaluation loop (1000-94 stages) processes all the remaining features until the end. This split strategy ensures that the first loop is usually the most time consuming part of the code. The reason of this behavior is due

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2019.2952335, IEEE Design and Test

to the fact the most discriminant LBP features are stored in the first 94 stages, which yield the early rejection of non-face regions.

Based on previous analysis not shown in the paper, we decided to only accelerate in hardware the first above mentioned loop due to the resource limitations of our platform. This is easily accomplished with our Nanos++ runtime as it can correctly orchestrate the execution of the task that implements the second loop based on the results of the task that implements the first one, even if every one of them is executed in a different resource (the FPGA and the SMP respectively).

All the experiments have been done in a Zynq-7000 (Zedboard) with a 7020 device at 200Mhz using a 680x383 image with 40 faces. OmpSs@FPGA has been used for rapid prototyping of several versions of the first loop partition of the cascade. OmpSs@FPGA also allows exploiting heterogeneous parallel execution of tasks that are specified `fpga` with those specified `smp`. However, in the experiments shown in this section we have forced a serialization of the FPGA task execution to measure the speedup achieved only in the acceleration of the first loop using the OmpSs@FPGA ecosystem. This is achieved annotating the function with the first loop partition using `target device(fpga)`, and calling `taskwait` immediately after each function call. Therefore, even more improvements are expected when exploiting the task-based parallelism of the code.

We have implemented several versions of the first loop partition that fully utilize all the FPGA resources available in the board. Note that Vivado HLS suffers from optimization difficulties when a loop has a `break` statement, which is a key software optimization. Our proposals explore those optimization issues to help the HLS compiler to exploit the FPGA resources.

Figure 4 shows the execution time for the second loop partition (constant time on an ARM core), and for nine variants of four different configurations of the first loop partition running on the FPGA (x-axis):

1) *Seq. w/ Early Rejection*. Sequential execution on an ARM core, as the baseline.
2) $\{3, 4, 6, 12\}$+*Early Rejection*. The first loop partition is split into a first group (1G in the figure) of iterations with $n \in \{3, 4, 6, 12\}$ iterations without early rejection (`break`), and the remaining iterations with early rejection.
3) $\{3, 4, 6\}$+*blocking*. The first loop partition is split into a first group as before with $n \in \{3, 4, 6\}$, and the remaining iterations are organized in blocks of $\{3, 4, 6\}$ iterations, with only early rejection between blocks. In addition, the blocks of loops are only applied to those sliding windows that have not been rejected in the first level.
4) *Blocking*. All iterations are organized in one group of blocks of 4 iterations. There is only early rejection between blocks.

The rapid prototyping done with OmpSs@FPGA allows to evaluate several versions in a short period of time. In the case of the non-blocked version, the bigger the first group, the worse the performance, since those number of iterations are always performed, as there is no early rejection. On the other hand, the remaining iterations with early rejection are not optimized by the HLS compiler, and thus it slowdowns the kernel.

The best performance is obtained by the $\{3, 4\}$+*blocking* configurations, reaching a $1.67\times$ speedup for the 4+*blocking* case when compared to the sequential version. This implementation allows Vivado HLS to perform pipelining and loop unroll (annotated with Vivado HLS pragmas in the implementation) to fully exploit the resources of the target FPGA in both groups of the first partition. In contrast, both the *blocking* and the early rejection configurations yield serious optimization problems to the HLS compilers, thus significantly degrading performance.

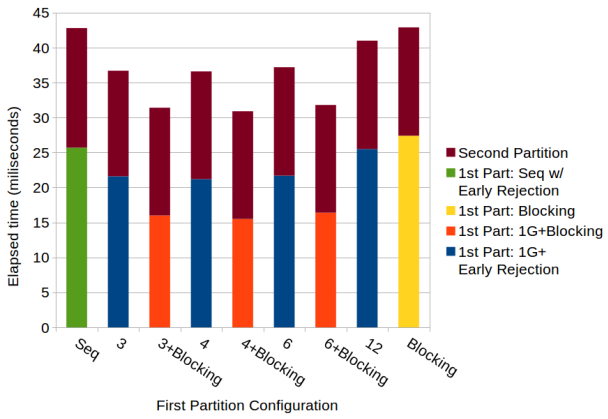

Fig. 4. OmpSs@FPGA Software/Hardware Co-design.

B. Kernel from the SHL scenario

The algorithm implemented on the *anisotropic smoothing module* consists of a loop with *m* iterations in which all image pixels are read and written several times. In all the iterations, the algorithm applies a stencil computation on each pixel, using the neighboring pixels.

This module processes a set of regions of interest (ROIs) of the input video frames defined by the iris recognition algorithm. In the implemented SLH application, the maximum area of these ROIs is 260x260 pixels. This limited size enables to develop an OmpSs@FPGA accelerator in which all the pixels of the ROI are saved in FPGA block RAM resources. In order to implement an FPGA accelerator for this module, a code refactoring has been done to meet the constraints introduced by the Vivado HLS compiler. The two main actions in this process have been to adapt the maximum size of the images to be processed in the accelerator, to fit the BRAM blocks in the FPGA, and use static (instead of dynamic) allocation of the variables involved.

The OmpSs annotations added to the anisotropic smoothing code define an OmpSs task that includes all the operations of the anisotropic smoothing algorithm, and sets the FPGA as the target device for such task. Further low-level optimizations were explored by annotating the code with HLS directives. Note that the OmpSs@FPGA environment allows to include

the HLS directives directly in the original source code, allowing the programmer to have one single unified source code for programming heterogeneous environments and consequently simplifying the task significantly.

Table I shows the execution time of the module on the ZC706 board. We present both the sequential and parallelized execution times. The obtained speedup when executing the synthesized code annotated with OmpSs on the FPGA was $8.4\times$.

TABLE I
EVALUATION OF THE ANISOTROPIC SMOOTHING MODULE.

Input pixels	Seq. Exec. time (s.)	OmpSs@FPGA Exec. time (s.)	Speedup
260x260	2.11	0.25	8.4x

VII. RELATED WORK

In the last few years CPS domain has gained huge importance, and as a consequence a lot of work has been done by both academia and industry. Shi et al. [9] have a general survey on CPS. In addition, Lee et al. [10] provided a general framework for using CPS in the manufacturing industry.

AXIOM aims to provide a generic programming model which can work with its high-speed interconnect subsystem on multiple platforms together with its full stack of software as well as proper hardware support. To do so it uses OmpSs, a forerunner of OpenMP that has been developed at BSC to experiment with new features and analyze the ones worth of being introduced in the standard.

In order to integrate heterogeneous execution of the same applications over processors and FPGA fabric, OmpSs@FPGA is a key point in the project. To the best of our knowledge OmpSs@FPGA [11] is the first successful attempt to implement hardware accelerators and trace them [12] from high-level directives in a total transparent way. Other tools try to reduce the FPGA programmability problem by offering the possibility of generating HDL code from C or C-like languages like ROCCC [13] or generating systems with an embedded soft processor connected to the generated hardware accelerators like LegUp [14] and C2H tool [15]. However, with the new SMP/FPGA SoCs, new strategies are required in order to exploit those current heterogeneous *and* parallel platforms. Our ecosystem also covers runtime support for parallel execution of heterogeneous tasks on those SoCs, unlike other.

VIII. CONCLUSION

This paper shows the implementation results of two real applications in an embedded heterogeneous system using the OmpSs@FPGA framework.

OmpSs@FPGA allows the programmers to focus on functionality and performance by alleviating the burden of generating and managing hardware accelerators and copies. With this framework, complex applications can be adapted to low-power embedded systems and fast functionality/performance explorations can be performed.

As the IoT ecosystem evolves, more functionalities are demanded from connected systems. Also, new more complex processors that include multi-cores and accelerators are being introduced. OmpSs@FPGA leverages directive-based programming to overcome this issues and provide performance at a reasonable programming effort.

ACKNOWLEDGMENT

This work is partially supported by the European Union H2020 program through the AXIOM project (grant ICT-01-2014 GA 645496), and HiPEAC (GA 687698), by the Spanish Government through Programa Severo Ochoa (SEV-2015-0493), by the Spanish Ministry of Science and Technology (TIN2015-65316-P) and the Departament d'Innovació, Universitats i Empresa de la Generalitat de Catalunya, under project MPEXPAR: Models de Programació i Entorns d'Execució Paral·lels (2014-SGR-1051).

REFERENCES

[1] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas, "Ompss: a proposal for programming heterogeneous multi-core architectures," *Parallel Processing Letters*, vol. 21, no. 2, pp. 173–193, 2011. [Online]. Available: https://doi.org/10.1142/S0129626411000151

[2] S. Mazumdar et.al., "AXIOM: A Hardware-Software Platform for Cyber Physical Systems," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 539–546.

[3] D. Theodoropoulos, N. Alachiotis, and D. N. Pnevmatikatos, "Multi-fpga evaluation platform for disaggregated computing," in *25th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2017, Napa, CA, USA, April 30 - May 2, 2017*, 2017, p. 193. [Online]. Available: https://doi.org/10.1109/FCCM.2017.20

[4] F. Sainz, S. Mateo, V. Beltran, J. L. Bosque, X. Martorell, and E. Ayguadé, "Leveraging ompss to exploit hardware accelerators," in *26th IEEE International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2014, Paris, France, October 22-24, 2014*, 2014, pp. 112–119. [Online]. Available: https://doi.org/10.1109/SBAC-PAD.2014.26

[5] Alvarez *et al.*, "The AXIOM software layers," *ELSEVIER Microprocessors and Microsys.*, vol. 47, Part B, pp. 262–277, 2016.

[6] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[7] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," *IEEE Custom Integrated Circuits Conference*, 2017.

[8] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[9] Jianhua Shi, Jiafu Wan, Hehua Yan, Hehua Yan, "A survey of cyber-physical systems," in *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*. IEEE, 2011, pp. 1–6.

[10] J. Lee, B. Bagheri, and H.-A. Kao, "A Cyber-Physical Systems Architecture for Industry 4.0-based Manufacturing Systems," *Manufacturing Letters*, vol. 3, pp. 18 – 23, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S221384631400025X

[11] A. Filgueras, E. Gil, D. Jiménez-González, C. Álvarez, X. Martorell, J. Langer, J. Noguera, and K. Vissers, "Ompss@zynq all-programmable soc ecosystem," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, ser. FPGA '14. New York, NY, USA: ACM, 2014, pp. 137–146. [Online]. Available: http://doi.acm.org/10.1145/2554688.2554777

[12] G. Llort, A. Filgueras, D. Jiménez-González, H. Servat, X. Teruel, E. Mercadal, C. Álvarez, J. Giménez, X. Martorell, E. Ayguade, and J. Labarta, "The Secrets of the Accelerators Unveiled: Tracing Heterogeneous Executions through OMPT," *12th International Workshop on OpenMP, IWOMP*, pp. 217 – 236, 2016.

[13] W. A. Najjar and J. R. Villarreal, "Fpga code accelerators - the compiler perspective," in *DAC*, 2013, p. 141.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2019.2952335, IEEE Design and Test

[14] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "Legup: High-level synthesis for fpga-based processor/accelerator systems," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 33–36. [Online]. Available: http://doi.acm.org/10.1145/1950413.1950423

[15] Altera, Corp., *Nios II C2H Compiler User Guide*, 2009.

Antonio Filgueras received a degree in computer science at Universitat Politècnica de Catalunya - BarcelonaTech (UPC). Currently working at the Programming Models group of Barcelona Supercomputing Center (BSC-CNS).

Miquel Vidal finished his MSc in High-Performance Computing at UPC-BarcelonaTech in 2017 and is currently working at the Programming Models group at BSC-CNS.

Marc Mateu got his BSc in Computer Engineering at the UPC-BarcelonaTech in 2017, and he is currently a Master Student at the BSC-CNS.

Daniel Jiménez-González is a tenured assistant professor at the Computer Architecture Department at UPC-BarcelonaTech, and is an associated researcher at the BSC-CNS.

Carlos Álvarez is a tenured assistant professor at the Computer Architecture Department at UPC-BarcelonaTech, and is an associated researcher at the BSC-CNS.

Xavier Martorell is an associate professor of computer science at the UPC-BarcelonaTech, and is leading the Parallel Programming Models group at BSC-CNS.

Eduard Ayguadé is full professor on computer science at UPC-BarcelonaTech, and associate director of research of the Computer Sciences Department at BSC-CNS.

Dimitrios Theodoropoulos obtained his PhD from the Computer Engineering department of the Delft University of Technology, the Netherlands. From 2011, he works as a post-doc researcher at the Foundation for Research and Technology - Hellas (FORTH) in Greece, in the fields of Embedded Systems, and Reconfigurable computing.

Dionisios Pnevmatikatos is a Professor at the School of ECE, Technical University of Crete, and a research associate at FORTH-ICS. His research interests include computer architecture, reconfigurable computing and highly efficient, accelerated, heterogeneous parallel/rack-scale systems.

Paolo Gai , PhD, is CEO and founder of EvidenceSRL, a SME working on operating systems, code generation, and model based design. His research interests include architectures for real-time embedded control systems, multicores, and parallel systems.

Stefano Garzarella is a Software Engineer with a strong experience in kernel and user space programming. He graduated in Computer Engineering at the University of Pisa in 2014. Currently, he works for EvidenceSRL on Linux embedded systems.

David Oro obtained his M.S. degree in Computer Science from UPC-BarcelonaTech. He currently works in the Research Department of Herta Security.

Javier Hernando received the M.S. and Ph.D. degrees in Telecommunication Engineering from UPC-BarcelonaTech. He is Full Professor and the Director of the UPC Research Center for Language and Speech (TALP).

Nicola Bettin obtained his M.S degree in Electronic Engineering at University of Bologna. He joined the electronic R&D dept. at Vimar Group in 2015. His main interests are FPGA and heterogeneous multi-core system-on-chip solutions.

Alberto Pomella has the degree in Electronic Engineering from Politecnico di Torino, specialization in computer science and industrial automation. Currently Electronics & Software R&D Manager at Vimar S.p.A., Standalone and Home and Building Automation products.

Marco Procaccini is a Ph.D. Student at the Department of Information Engineering and Mathematics, University of Siena, Italy. He graduated in Computer Science at the University of Salerno in 2016, Magna cum Laude.

Roberto Giorgi is an Associate Professor at University of Siena, Italy. He received his PhD in Computer Engineering and his Master in Electronics Engineering, Summa cum Laude from University of Pisa, Italy. He coordinated the AXIOM and TERAFLUX European projects.