AXIOM: A Scalable, Efficient and Reconfigurable Embedded Platform

Roberto Giorgi[1], Marco Procaccini[1] and Farnam Khalili[1],[2]

[1]Department of Information Engineering and Mathematics - University of Siena
[2]Department of Information Engineering - University of Florence
{giorgi,procaccini,khalili}@diism.unisi.it

***Abstract*—Cyber-Physical Systems (CPSs) are becoming widely used in every application that requires interaction between humans and the physical environment. People expect this interaction to happen in real-time and this creates pressure onto system designs due to the ever-higher demand for data processing in the shortest possible and predictable time. Additionally, easy programmability, energy efficiency, and modular scalability are also important to ensure these systems to become widespread. All these requirements push new scientific and technological challenges towards the engineering community. The AXIOM project (Agile, eXtensible, fast I/O Module), presented in this paper, introduces a new hardware-software platform for CPS, which can provide an easy parallel programming model and fast connectivity, in order to scale-up performance by adding multiple boards. The AXIOM platform consists of a custom board based on a Xilinx Zynq Ultrascale+ ZU9EG SoC including four 64-bit ARM cores, the Arduino socket and four high-speed (up to 18 Gbps) connectors on USB-C receptacles. By relying on this hardware, DF-Threads, a novel execution model based on dataflow modality, has been developed and tested. In this paper, we highlight some major conclusions of the AXIOM project, such as the gain in performance compared to other parallel programming models such as OpenMPI and Cilk.**

***Index Terms*—Cyber-Physical Systems, Reconfigurable Systems, FPGA Programming, Thread Level Parallelization, Energy Efficiency, Embedded Systems.**

I. Introduction

Nowadays, with the fast improvements in science, technology, and engineering, designers are gradually redefining the capabilities of computing systems around us to improve the so called "Embedded Intelligence" or "Smart Things". In fact, both "things" and people are becoming nodes of the same network, creating a Cyber Physical domain [1]. CPSs operate through intelligent interfaces to communicate via web and social media and interact with the environment. In our daily life, CPSs devices provide us with efficiency, flexibility such as in the case of smartphones, smart home and assisted/autonomous driving. Therefore, the growing number of applications have created a huge fragmentation in hardware platforms and software tools. Also, some of the main challenges in designing a CPS architecture are data management, proper software-hardware integration, real-time management and hardware specialization. By building upon successful examples of design-for-simplicity, like in the case of Arduino [2] and UDOO [3], the AXIOM platform (Agile, eXtensible, fast I/O Module) [4]–[11] aims to provide a complete and general software development suite for easily mapping applications into multi-board processing systems.

According to known road maps for future systems, the crucial problems for a broader deployment of scalable embedded systems are easy programmability, and inexpensive ways to build a system based on the simpler components. The AXIOM project has defined a simple but powerful architecture that can possibly be deployed in CPS, since it includes not only the conventional embedded components but also the possibility to easily build CPS by using one, two or more boards, without changing programming model.

A Single Board Computer (SBC), named "AXIOM-Board", has been developed at the beginning of the 2017, Figure 1, to build up a heterogeneous system, which could be able to combine ARM cores and enough programmable logic (FPGA) for significant acceleration, providing a platform that can be suitable for wide range of scenarios like Artificial Intelligence, Smart Home Living, Smart Video Surveillance, just to name a few.

In this paper, we will draw some conclusions of the main achievements of the AXIOM project, such as the programming model, based on a modified version of OmpSs [12] and the Data-Flow Threads (DF-Threads) execution model [13]–[24]. We describe the fast link interconnect, named "AXIOM-Link", through which is possible to connect multiple boards using inexpensive cables, such as USB-C ones, while reaching up to 18-Gbps for each channel.

II. The 64-bit AXIOM platform

One of the main goals of the AXIOM project was to design the hardware/software layers for multi-core, multi-board and heterogeneous system that has been envisioned by the project partners in order to fulfill the needs of future Cyber-Physical Systems (CPS). In this respect, the partners (BSC, EVIDENCE, FORTH, HERTA, SECO, University of Siena, VIMAR) identified use case scenarios, analyzed the AXIOM concept against possible exploitation paths, evaluated the AXIOM board to assess its capabilities. The

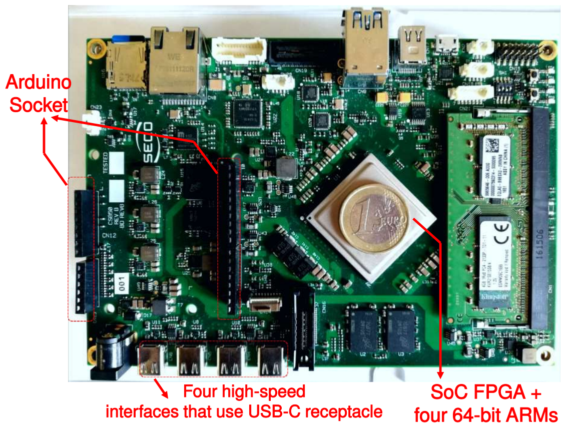
Arduino Socket
Four high-speed interfaces that use USB-C receptacle
SoC FPGA + four 64-bit ARMs

Fig. 1: The AXIOM board based on a MPSoC Zynq Ultrascale+ (ZU9EG platform, expandable DDR4 memory up to 32 GiB, with four 2-lane 10Gbps gigabit transceivers that use USB-C cables, and an Arduino socket.

inferred information has been translated into the definition of the AXIOM architecture (i.e., the need for an SoC with high-speed interconnects and FPGA), its external interface and its functional requirements.

For instance, modular scalability is enabled by a high throughput and low latency interconnect and the possibility to interface the applications directly to such high speed interconnect via reconfigurable hardware. Moreover, the Arduino UNO socket permits the use of a large set of tested so-called “shield” containing, e.g., relays, sensors and actuators. The Arduino software AVR-binary compatibility is ensured by a custom AVR soft-IP. This led to the choice of using the Xilinx MP-SoC Zynq Ultrascale+ (ZU9EG) platform [25]. Also, this choice opens the possibility to reach a design that is capable of being interfaced to the physical world and be used in smart applications where critical operation could be offloaded to the FPGA. The FPGA also provides a greater energy efficiency compared to executing the same function in software [26] and an appropriate substrate for the integration of our key features by providing customized and reconfigurable acceleration.

Therefore the design of the AXIOM hardware and software has been driven by the following pillars:

1) MP-SoC FPGA, i.e., The combination of large Programmable Logic (PL) with the ARM-based General Purpose Processors (GPPs), to support the Operating System (OS) and for running tasks that make little sense on the other accelerators,
2) Open-source software stack for a broader adoption,
3) Lower-Level Thread Scheduler for a higher predictability,
4) High-speed, inexpensive interconnects managed by an efficient Network Interface (NI) [27],
5) Efficient interfaces for the Cyber-Physical world, such as Arduino [2] connectors (to be able to interface with sensors and actuators), USB, Ethernet.

A. Hardware Specification

For the sake of completeness, we briefly recall here the main capabilities of our platform: the Xilinx ZU9EG platform [25] includes four 64-bit quad ARM Cortex-A53 General Purpose Processors (GPPs) working at a frequency of up to 1.5GHz; 32KB L1 Cache and 1MB L2 Cache. This can support a number of activities such as the OS (or system tasks), but also whenever there is a sequential task that invokes Instruction Level Parallelism (ILP) rather than other forms of acceleration. Moreover, there is a Dual-Core ARM Cortex-R5 processing unit specialized for Real-Time tasks, working at a frequency of up to 600Mhz; 32KB L1 Cache and 128KB of tightly coupled memory for each core. The processors are encapsulated as part of the processing system (PS) as well as general purpose interfaces such as: two UART, two full-duplex SPI, two full CAN 2.0B-compliant CAN, two USB, four 10/100/1000 tri-speed Ethernet, two I2C, ARM Mali-400 GPU, a display Port, DDR4 Controller, 17-channel 10-bit ADC and up to 128 GPIOs. Furthermore, the PL covers up to approximately 300K LUTs, 32 Mb Block-RAMs, up to 2,520 DSP slices, and up to 24 bidirectional gigabit transceivers with maximum 16.3 Gbps throughput. These transceivers are exposed to the physical world through the USB-C receptacle (using a custom AXIOM protocol, not the USB-C protocol), which is more easy to be used due to its two-fold rotationally-symmetrical connector. Moreover, the board has a 250MHz trace port which has been used also in other projects such as the H2020 HERCULES project [28].

B. Soft-IP: DF-Threads Scheduler

Some of the functions cannot be performed in software as they have an execution overhead which is too large.

The DF-Threads Scheduler serves to offload the management of the thread descriptors either locally or across the boards. For example, a load balancer unit is responsible to distribute the thread to other boards when local resources finish. In order to bypass the software stack, this scheduler is capable of dispatching the descriptors directly to NI.

In order to have an efficient and scalable execution and movement of threads across the AXIOM board, a low-level finer grained data distribution technique based upon the DataFlow-Threads (DF-Threads) [13], [14], [19] modality has been adopted. DF-Threads make possible to perform a more predictable real-time execution since the input data of a thread is made available before the thread execution, so the time to execute a thread can be estimated very precisely.

The DF-Thread memory model [18] makes it possible to manage different communication patterns such as producer-consumer (1-to-1 or N-to-1) but also generalized sharing of mutable data such as in patterns 1-to-N or N-to-N [14].

C. Soft-IP: Network Interface (NI)

In order to avoid the overheads and costs of proprietary existing communication protocols a NI has been designed and implemented on the PL [27]. This permits to achieve an efficient and scalable program execution as well as a seamless interconnection of systems spanning multiple boards [11]. Such connectivity permits users to expand and scale-up their system by interconnecting more boards in a flexible and low-cost manner, without the need for expensive particular cables, connectors or external switches.

AXIOM board has four bi-directional links providing different network topologies such as ring, torus and 2D-mesh and etc. The AXIOM routing algorithm is based on the store-and-forward packet transmission with virtual circuits (VCs). In order to fill up the routing tables by dedicated node IDs, a discovery process is initiated at power-up by the master node of the network. As such all the packets will be transceived through the physical links based on corresponding information stored in the routing table.

III. AXIOM Software Stack

A simplified view of the main components running on the PS are represented on the left side of Figure 2. In particular the purpose of these components is:

i) AXIOM APPLICATIONS, written in OmpSs, including the general runtime libraries, OmpSs library (Nanos++);
ii) AXIOM USER API, which includes AXIOM LIBRARIES and ALLOCATOR; the API is exposed to the OmpSs library, not directly to the user, and serve to manage from the user space the hardware resources, i.e., DF-Threads, NI and Memory;
iii) AXIOM DRIVERS, these are the kernel side software components to manage hardware;

In the following discussed each of those components in details.

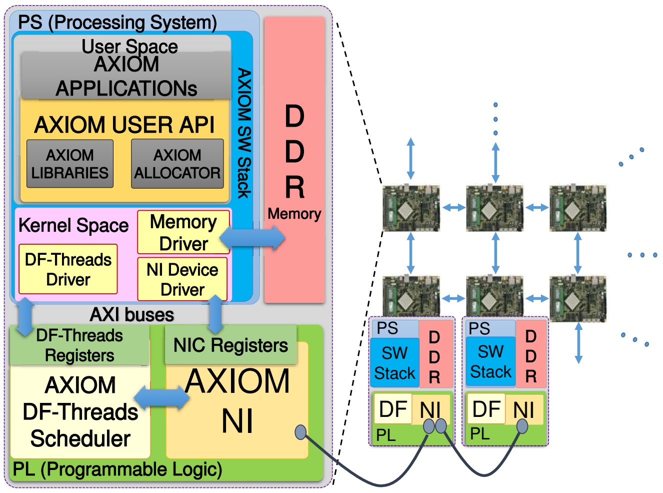

Fig. 2: The main components running on the PS and the Soft-IPs in the PL. On the right, a possible topology of an AXIOM cluster (2D-mesh). PS: Processing System, PL: Programmable Logic.

A. *AXIOM Applications*

The recent success of Single Board Computers (SBCs) such as the UDOO [3], and RaspberryPi [29], further highlighted the benefits of using open-source software, in order to simplify the maintenance of complex software stacks and enlarge the user communities. In AXIOM, we decided to rely on an Ubuntu distribution and a standard compilation tool-chain offered on such distribution. In order to parallelize the execution one of the most accepted paradigms is nowadays OpenMP [30]; we experimented with OmpSs [31], [32], which is now converging into the OpenMP standard and future versions of OpenMP are likely to include almost all the OmpSs features. OmpSs offer an easy way (via "pragmas") to identify the code to be parallelized onto specific targets such as a cluster, a GPU or an FPGA. Therefore, OmpSs is offering efficient parallelization techniques, which are familiar to the HPC programmer (i.e., OpenMP-like), to the Embedded System community. In this respect, OmpSs provides a first level of data distribution on a heterogeneous system.

In order to distribute tasks across the cluster, the programmer needs to specify the task plain in a higher-level code (like C++), and the OmpSs automatically allows the runtime system to spawn tasks across the remote notes. OmpSs permits parallelizing applications on the AXIOM cluster and spawn workloads on the FPGAs. OmpSs parallelizes the tasks in two layers: i) parallelization on the available ARM processor residing on PS part, ii) expressed the tasks into the PL. The OmpSs programming model is based on two main components: i) The Mercurium compiler [33], takes the source code and understands the OmpSs directives to transform the code to run on heterogeneous platforms , ii) The Nanos++ runtime system [12], which is the responsible to manage and schedule parallel tasks, transferring the data needed to/from the accelerators when needed. The OmpSs programming model provides a higher-level tasks definition including OpenCL, CUDA, C or C++ capable of being converted to the machine language used in GPUs or to the bitstream to configure FPGAs. Furthermore, the OmpSs provides runtime supports for the communications within a cluster of a DSM machine.

B. *AXIOM User API, Libraries and Allocator*

The AXIOM software architecture [6], [7] includes the techniques used for managing the memory in the cluster, the node interconnections, and the software stack used to manage the cluster, which is in turn composed of the device driver and the libraries for board-to-board communication and memory allocation.

The AXIOM user API offers the primitives to manage DF-Threads, send/receive of "raw" messages, "RDMA" messages and "long" messages [4] through the NI, and also the allocation primitives for both shared (meaning for more threads) and private (meaning for a single thread) allocations. It is based on dynamic libraries, which in turn use the IOCTL API and mmap facility to interface with the Kernel.

In addition, this API handles data passing and notifications between kernel and user space. In the user space, a set of libraries and daemons are used to service to the AXIOM application user space. As such, AXIOM allocator is responsible to handle a part of the memory of each board in the AXIOM cluster to support RDMA transactions provided by AXIOM NI as well as reserving a dedicated range of contiguous physical memory mapped to the various processes composing an AXIOM application.

C. AXIOM Drivers

The AXIOM (Kernel) Drivers take care of lower-level allocation of memory, making sure that the memory is consistent, also across boards, implement the IOCTL calls, manage the internal data structures such as software queues, handshaking, filling descriptors, buffering. The Soft-IPs are made visible to the applications through a set of registers and memory-mapped I/O.

IV. Methodology of Design and Evaluation

The methodology used in AXIOM to develop the most advanced concepts, which are related to the DF-Threads execution, consists in a co-design approach based on the COTSon framework [9], [34] and the Xilinx Vivado HLS tools [25].

COTSon is useful for the rapid prototyping of functionalities: we can test the functional behavior in the full-system execution, thus observing and measuring not only the impact of the application but also the impact of the OS activities, the runtime, and the libraries. Still on the COTSon simulator is possible to define the desired architecture and its timing: as a simple example, in the behavioral part one could model a cache as a simple hit probability based on a random variable ("OPERATION BEHAVIOR" in (Figure 3), while in the timing part the cache is defined in terms of a precise architecture made of tags, data, valid bit and the time to access them. This permits to identify the appropriate architecture and verify if the Key Performance Indicators/Metrics (KPIs) are met ("OPERATION TIMING" in Figure 3).

In addition, the simulation tool-chain has been augmented with tools for the Design Space Exploration [9], [35]–[37] (DSE) such as:

1) MYDSE tool: to launch a set of simulations (one for each design point);
2) GTCOLLECT tool: to collect the data in a database so that data analytics can be done;
3) GTGRAPH tool: to present in a graphical way a set of performance metrics and generate experiment reports.

In particular the MYDSE tool enables the user to:

i) Specify a DSE experiment through a simple configuration file;
ii) Automatically, distribute the simulations across a number of hosts;
iii) Manage automatically cases when a simulations fails or get stuck after a given time threshold by killing the simulation and all related processes;
iv) Properly use the same binary across multiple hosts with different GLIBC libraries and compilation tools binaries (depending on the library and compilation tools the simulation results may differ due to a different benchmark binary that get pulled into the guest);
v) Collect in an ordered way the several files from a single simulation point and from several simulation points belonging to the same experiment;
vi) Monitor and control the simulation loop;
vii) Interpret user formulas to extract aggregated information from several basic statistics (e.g. calculating the miss rate starting from the read miss and write miss);
viii) Automatically try to re-execute the simulations that eventually fail;
ix) Insert the code for marking the Region of Interest (ROI);
x) Parse and updating the configuration files of a simulation starting from a given template;
xi) Support different execution models such as DF-Threads, the standard one or others;

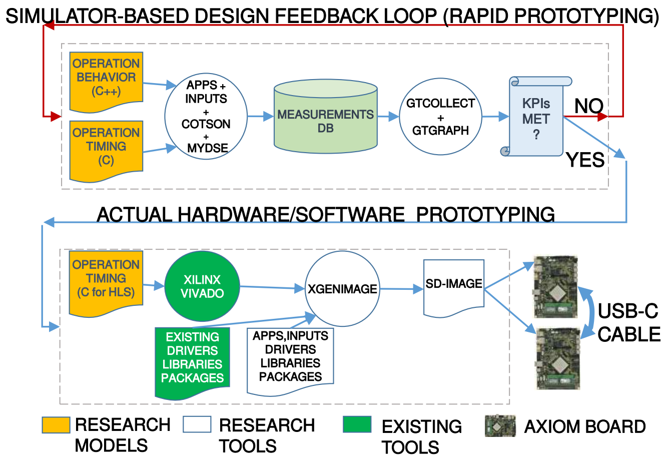
SIMULATOR-BASED DESIGN FEEDBACK LOOP (RAPID PROTOTYPING)
OPERATION BEHAVIOR (C++)
OPERATION TIMING (C)
APPS + INPUTS + COTSON + MYDSE
MEASUREMENTS DB
GTCOLLECT + GTGRAPH
KPIs MET ?
NO
YES
ACTUAL HARDWARE/SOFTWARE PROTOTYPING
OPERATION TIMING (C for HLS)
XILINX VIVADO
XGENIMAGE
SD-IMAGE
EXISTING DRIVERS LIBRARIES PACKAGES
APPS,INPUTS DRIVERS LIBRARIES PACKAGES
USB-C CABLE
RESEARCH MODELS
RESEARCH TOOLS
EXISTING TOOLS
AXIOM BOARD

Fig. 3: The design and test methodology of AXIOM involved a mix of simulation (via the COTSon simulator and other custom tools) and FPGA-prototyping (via our custom AXIOM board and hardware synthesis tools (like Vivado HLS).

In Figure 3, a simplified design flow is illustrated. In our case the tool-flow supported the design of the DF-Threads execution model. Once the KPIs are met, the "OPERATION TIMING" is translated into Vivado-HLS (the Xilinx tool for High Level Synthesis) so that it can be tested with actual hardware constraints on the FPGA. On this side, it has been necessary also to build the tools for generating a full system image ("SD-IMAGE", since it is typically stored on an SD-card, which can be directly plugged-in the board for booting it, programming the FPGA and running the applications). The "XGENIMAGE" tool, permits the configuration of the necessary parameters, for the Board-Support-Package, for the "Device-Tree", which describes the system peripherals and soft-IPs, and includes the needed additional libraries and drivers (e.g., to pilot the soft-IPs) or system level configuration (e.g., specific memory mapping for the applications).

The generated IP is then put besides other AXIOM related IPs, such as the NI, the AVR and other accelerators.

V. Evaluations and Results

During the AXIOM project, we analyzed two main real life applications, Smart Video Surveillance(SVS) and Smart Home Living (SHL). These applications are very computational demanding, since they require to analyze a huge number of scenes coming from multiple cameras located into airports, home, hotels or shopping malls. After thorough analysis on how to improve the performance of such case-studies, we figured out that the Matrix Multiplication kernel has a central role into the computation.
In this section, we want to analyze the performance of the DF-Threads, comparing it with the well-known programming model like OpenMPI [38] and Cilk [39]. As we described into the section IV, we use COTSon simulator as a testing environment and we present some of the obtained results.

A. *Matrix Multiplication benchmark*

Several versions of classic MM algorithm exist and we selected the Block Matrix Multiply algorithm(BMM), where a matrix is partitioned in multiple sub-matrices, or blocks, according to the block size that is set. The core of the BMM algorithm is the classical 3 nested loops and we decided to use square matrix of size 512x512 with 8 as block size for our evaluation. Also, we focused our measurement only on the computational region avoiding the tasks that deal with the preparation of data and the correctness of the output, because they are irrelevant for a fair comparison [40].

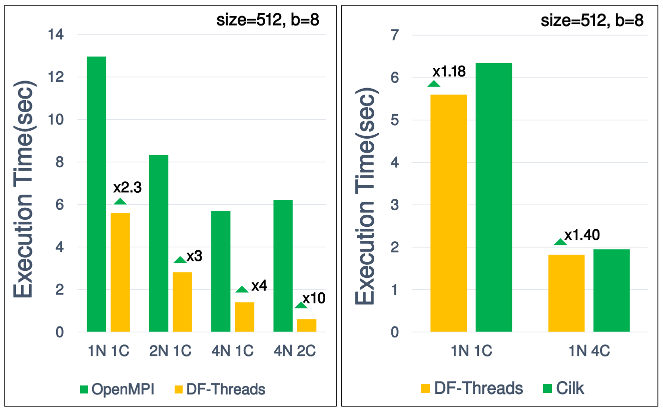

Fig. 4: Execution time comparison between DF-Threads, OpenMPI (left), Cilk(right) . The performance scale up increasing the number of nodes and core in comparison of both OpenMPI and Cilk.

B. *Results*

The Execution Time in seconds is presented in Figure 4, where we direct compare DF-Threads, OpenMPI and Cilk.

Cilk is optimized of the multi-core platform, we are able to compare it with DF-Threads just increasing the number of cores. Results show that the execution time of DF-Threads slightly outperform Cilk. In the comparison of OpenMPI, we increased both the number of cores and nodes obtaining better results in every configuration, with the maximum boost

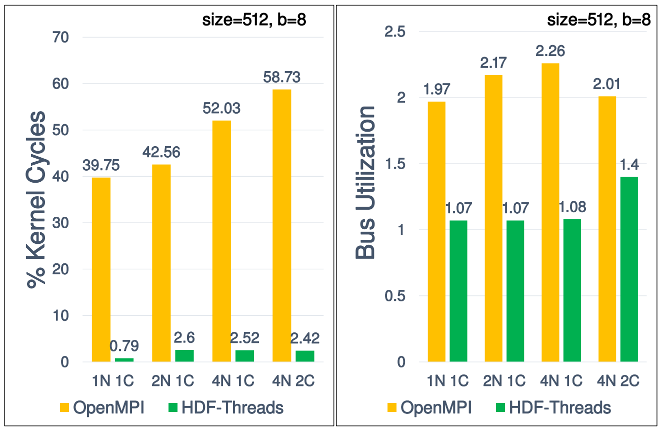

Fig. 5: Comparison of the Kernel cycles and bus utilization between DF-Threads and OpenMPI. As we increase the number of core and nodes the percentage of kernel cycles increase in OpenMPI, while DF-Thread remains on low values in every condition. Also, the overall bus utilization is lower in DF-Threads than OpenMPI.

of the performance in the configuration with 4 nodes and 2 cores. Thanks to the COTSon simulator we can analyze key metrics like the percentage of the kernel utilization and the bus utilization (Figure 4), showing that the OpenMPI has a huge kernel activity which increases proportionally with the number of the nodes. Moreover, DF-Threads execution model demonstrates a better bus utilization, proving the minimization of the data exchange.

VI. Conclusions

In this paper, we presented some of the main achievements of the AXIOM project. In particular, we highlighted the realization of our own manufactured board (named AXIOM board) and a complete software stack of more than one million lines of source code entirely made available as open-source software at https://git.axiom-project.eu/ .

We presented the tool-chain, which is based on both a full system simulator (a modified version of HP-Labs COTSon, also made public) and standard synthesizer for the Xilinx Ultrascale+ platform.

Users of this platform can benefit from performance scalability by just interconnecting more boards via USB-C cables without the need of any other components (like external switches). The performance is obtained thanks to a novel execution model based on DF-Threads. DF-threads are distributed thanks to availability of a hardware scheduler, which can directly exchange the hand-shake information and data through the high-speed interconnection (which can achieve up to 18 Gpbs per channel).

In real use cases like smart video surveillance and smart home, we observed the need to offload part of the application on the FPGA as well as distributing large workloads on several AXIOM boards. We compared the performance of the DF-Threads against OpenMPI and Cilk, and we found out that DF-Threads can outperform OpenMPI by a factor of 10x in the case of four-boards (nodes).

VII. Acknowledgement

The authors would like to thank the anonymous reviewers for their feedback and the AXIOM project partners for the interactive discussions and lessons learned during the project duration: Barcelona Supercomputing Center (BSC) [32], EVIDENCE (s.r.l) [41], FORTH [42], HERTA [43], SECO (s.r.l) [44], University of Siena and VIMAR [45].
This work has been partially supported by the European Commission under the AXIOM H2020 project (grant ICT-01-2014 GA 645496) and HiPEAC (id. 779656).

References

[1] E. Geisberger and M. Broy, *Living in a networked world: Integrated research agenda Cyber-Physical Systems (agendaCPS)*. Herbert Utz Verlag, 2015.

[2] M. Banzi, *Getting Started with Arduino*. Sebastopol, CA: Make Books - O'Reilly Media, 2008.

[3] A. Rizzo, G. Burresi, F. Montefoschi, M. Caporali, and R. Giorgi, "Making iot with udoo," *Interaction Design and Architecture(s)*, vol. 1, pp. 95–112, Dec. 2016.

[4] D. Theodoropoulos, S. Mazumdar, E. Ayguade, N. Bettin, J. Bueno, S. Ermini, A. Filgueras, D. Jimenez-Gonzalez, C. Alvarez Martinez, X. Martorell, F. Montefoschi, D. Oro, D. Pnevmatikatos, A. Rizzo, P. Gai, S. Garzarella, B. Morelli, A. Pomella, and R. Giorgi, "The axiom platform for next-generation cyber physical systems," *ELSEVIER Microprocessors and Microsystems*, pp. 540–555, 2017.

[5] R. Giorgi, "Scalable embedded systems: Towards the convergence of high-performance and embedded computing," in *Proc. 13th IEEE/IFIP Int.l Conf. on Embedded and Ubiquitous Computing (EUC 2015)*, pp. 148–153, Oct. 2015.

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

[7] C. Alvarez *et al.*, "The AXIOM software layers," in *IEEE Proc. 18th EUROMICRO-DSD*, pp. 117–124, Aug. 2015.

[8] R. Giorgi, "AXIOM: A 64-bit reconfigurable hardware/software platform for scalable embedded computing," in *6th Mediterranean Conf. on Embedded Computing (MECO)*, pp. 113–116, June 2017.

[9] R. Giorgi, N. Bettin, P. Gai, X. Martorell, and A. Rizzo, *AXIOM: A Flexible Platform for the Smart Home*, ch. 3, pp. 57–74. Cham: Springer Int.l Pub., 2016.

[10] D. Theodoropoulos *et al.*, "The AXIOM project (agile, extensible, fast i/o module)," in *IEEE Proc. 15th Int.l Conf. on Embedded Computer Systems: Architecture, MOdeling and Simul.*, pp. 262–269, July 2015.

[11] R. Giorgi, S. Mazumdar, S. Viola, P. Gai, S. Garzarella, B. Morelli, D. Pnevmatikatos, D. Theodoropoulos, C. Alvarez, E. Ayguade, J. Bueno, A. Filgueras, D. Jimenez-Gonzalez, and X. Martorell, "Modeling multi-board communication in the axiom cyber-physical system," *Ada User Journal*, vol. 37, pp. 228–235, December 2016.

[12] 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. 02, pp. 173–193, 2011.

[13] R. Giorgi, "Scalable embedded computing through reconfigurable hardware: comparing df-threads, cilk, OpenMPI and jump," *ELSEVIER Microprocessors and Microsystems*, vol. 63, pp. 66–74, Aug. 2018.

[14] R. Giorgi and P. Faraboschi, "An introduction to DF-Threads and their execution model," in *IEEE MPP*, (Paris, France), pp. 60–65, Oct. 2014.

[15] L. Verdoscia and R. Giorgi, "A data-flow soft-core processor for accelerating scientific calculation on FPGAs," *Mathematical Problems in Engineering*, vol. 2016, pp. 1–21, Apr. 2016. article ID 3190234.

[16] N. Ho, A. Mondelli, A. Scionti, M. Solinas, A. Portero, and R. Giorgi, "Enhancing an x86_64 multi-core architecture with data-flow execution support," in *ACM Computing Frontiers*, pp. 1–2, May 2015.

[17] N. Ho, A. Portero, M. Solinas, A. Scionti, A. Mondelli, P. Faraboschi, and R. Giorgi, "Simulating a multi-core x86-64 architecture with hardware isa extension supporting a data-flow execution model," in *IEEE Proc. AIMS-2014*, (Madrid, Spain), pp. 264–269, Nov. 2014.

[18] R. Giorgi, "Exploring dataflow-based thread level parallelism in cyber-physical systems," in *Proc. ACM Int.l Conf. on Computing Frontiers*, CF '16, (New York, NY, USA), pp. 295–300, ACM, 2016.

[19] R. Giorgi and A. Scionti, "A scalable thread scheduling co-processor based on data-flow principles," *ELSEVIER Future Generation Computer Systems*, vol. 53, pp. 100–108, Dec. 2015.

[20] R. Giorgi, Z. Popovic, and N. Puzovic, "Implementing fine/medium grained tlp support in a many-core architecture," in *Proc. 9th Int.l Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2009*, (Samos, Greece), pp. 78–87, Springer, July 2009.

[21] R. Giorgi, Z. Popovic, and N. Puzovic, "Exploiting DMA to enable non-blocking execution in decoupled threaded architecture," in *Proc. IEEE Int.l Symp. on Parallel and Distributed Processing - MTAAP Multi-Threading Architectures and APplications*, (Rome, Italy), pp. 2197–2204, IEEE, May 2009.

[22] A. Portero, Z. Yu, and R. Giorgi, "Teraflux: Exploiting tera-device computing challenges," *ELSEVIER Procedia Computer Science*, vol. 7, pp. 146–147, 2011. Proc. 2nd European Future Technologies Conf. and Exhibition 2011 (FET 11).

[23] A. Portero, A. Scionti, Z. Yu, P. Faraboschi, C. Concatto, L. Carro, A. Garbade, S. Weis, T. Ungerer, and R. Giorgi, "Simulating the future kilo-x86-64 core processors and their infrastructure," in *45th Annual Simulation Symp. (ANSS12)*, (Orlando, FL), pp. 62–67, Mar 2012.

[24] A. Mondelli, N. Ho, A. Scionti, M. Solinas, A. Portero, and R. Giorgi, "Dataflow support in x86-64 multicore architectures through small hardware extensions," in *IEEE Proc. DSD*, pp. 526–529, August 2015.

[25] Xilinx Inc., "Xilinx UltraScale Architecture."

[26] L. Józwiak, "Embedded computing technology for highly-demanding cyber-physical systems," *IFAC-PapersOnLine*, vol. 48, no. 4, pp. 19 – 30, 2015. 13th IFAC and IEEE Conference on Programmable Devices and Embedded Systems.

[27] V. A. Lorentzos, *Efficient network interface design for low cost distributed systems*. Master thesis at Technical University of Crete, 2017.

[28] Hercules2020, "https://hercules2020.eu."

[29] The Raspberry Pi Foundation., "The Raspberry Pi Board."

[30] L. Dagum and R. Menon, "Openmp: An industry standard api for shared-memory programming," in *IEEE International Conference on Computational Science and Engineering*, pp. 46–55, Jan 1998.

[31] J. Bueno, L. Martinell, A. Duran, M. Farreras, X. Martorell, R. Badia, E. Ayguade, and J. Labarta, "Productive cluster programming with OmpSs," *Euro-Par Parallel Processing*, pp. 555–566, 2011.

[32] Barcelona Supercomputing Center (BSC), "https://www.bsc.es/."

[33] J. Balart, A. Duran, M. Gonzàlez, X. Martorell, E. Ayguadé, and J. Labarta, "Nanos mercurium: a research compiler for openmp," in *Proceedings of the European Workshop on OpenMP*, vol. 8, p. 56, 2004.

[34] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "COTSon: infrastructure for full system simulation," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, pp. 52–61, 2009.

[35] G. Palermo, C. Silvano, and V. Zaccaria, "Respir: A response surface-based pareto iterative refinement for application-specific design space exploration," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 28, no. 12, pp. 1816–1829, 2009.

[36] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "Multi-processor system-on-chip design space exploration based on multi-level modeling techniques," in *ICSAMOS*, pp. 118–124, IEEE, 2009.

[37] G. Mariani, A. Brankovic, G. Palermo, J. Jovic, V. Zaccaria, and C. Silvano, "A correlation-based design space exploration methodology for multi-processor systems-on-chip," in *DAC*, pp. 120–125, ACM, 2010.

[38] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proc., 11th European PVM/MPI Users' Group Meeting*, (Budapest, Hungary), pp. 97–104, September 2004.

[39] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: an efficient multithreaded runtime system," *SIGPLAN Not.*, vol. 30, no. 8, pp. 207–216, 1995.

[40] L. Verdoscia, R. Vaccaro, and R. Giorgi, "A matrix multiplier case study for an evaluation of a configurable dataflow-machine," in *ACM CF'15 - LP-EMS*, pp. 1–6, May 2015.

[41] EVIDENCE, s.r.l., "http://www.evidence.eu.com/."

[42] Foundation for Research and Technology - Hellas (FORTH), "https://www.forth.gr/."

[43] HERTA, Spain, "http://www.hertasecurity.com/en."

[44] SECO, s.r.l., "http://www.seco.com."

[45] VIMAR, Italy, "https://www.vimar.com," 1945.