Microprocessors and Microsystems 000 (2017) 1–16

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

The AXIOM platform for next-generation cyber physical systems

Dimitris Theodoropoulos[a,*], Somnath Mazumdar[b], Eduard Ayguade[c,f], Nicola Bettin[d], Javier Bueno[c], Sara Ermini[e], Antonio Filgueras[c], Daniel Jiménez-González[c,f], Carlos Álvarez Martínez[c,f], Xavier Martorell[c,f], Francesco Montefoschi[e], David Oro[g], Dionisis Pnevmatikatos[a,h], Antonio Rizzo[e], Paolo Gai[i], Stefano Garzarella[i], Bruno Morelli[i], Alberto Pomella[d], Roberto Giorgi[b]

[a] Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH) - Crete, Greece
[b] Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università degli Studi di Siena, Italy
[c] Barcelona Supercomputing Center (BSC), Barcelona, Spain
[d] VIMAR SpA Marostica, Italy
[e] Dipartimento di Scienze Sociali, Politiche e Cognitive, Università degli Studi di Siena, Italy
[f] Computer Architecture Department, Universitat Politècnica de Catalunya, Barcelona, Spain
[g] Herta Security Barcelona, Spain
[h] School of ECE, Technical University of Crete, Chania, Greece
[i] Evidence Srl

ARTICLE INFO

Article history:
Received 28 December 2016
Revised 24 May 2017
Accepted 29 May 2017
Available online xxx

Keywords:
Cyber-physical systems
Distributed shared memory
Programming model
Performance evaluation
Reconfigurable
Smart video surveillance
Smart home living

ABSTRACT

Cyber-Physical Systems (CPSs) are widely used in many applications that require interactions between humans and their physical environment. These systems usually integrate a set of hardware-software components for optimal application execution in terms of performance and energy consumption. The AXIOM project (Agile, eXtensible, fast I/O Module), presented in this paper, proposes a hardware-software platform for CPS coupled with an easy parallel programming model and sufficient connectivity so that the performance can scale-up by adding multiple boards. AXIOM supports a task-based programming model based on OmpSs and leverages a high-speed, inexpensive communication interface called AXIOM-Link. The board also tightly couples the CPU with reconfigurable resources to accelerate portions of the applications. As case studies, AXIOM uses smart video surveillance, and smart home living applications.

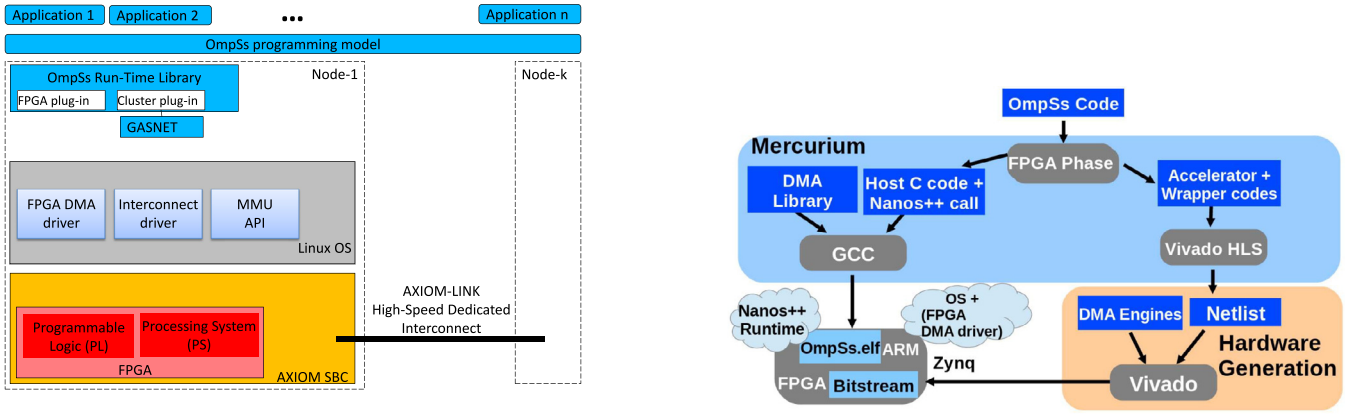
1. Introduction

"Cyber-physical systems (CPSs) integrate computation, communication, sensing, and actuation with physical systems to fulfill time-sensitive functions with varying degrees of interaction with the environment, including human interaction." [1]. A similar definition for CPS is "an integrated framework of a network of information processing, sensors and actuators" [2,3]. Such systems allow a close interaction not only system to system, but also with human-system or vice-versa, and are getting ever more pervasive in many daily life activities [4–6]. The CPS domain includes Internet of Things (IoT), smart homes, smart cities, or the smart grid. Everyday life is becoming increasingly dependent on CPS (e.g., smart video surveillance). Since 2008 CPS is a high priority research topic [7]. The noted challenges in designing a CPS architecture are infrastructural challenges, time management, data management (the data workflow), proper software-hardware integration (implementational challenges) and compliance with standards.

The AXIOM project (Agile, eXtensible, fast I/O Module) provides a general framework focusing on easily mapping applications to multi-board processing platforms [8,9]. Unlike other research efforts (such as CONTREX [10], DREAMS [11], EMC^2 [12], MultiPARTES [13]) that focus mainly on the mixed-criticality appli-

* Corresponding author.
E-mail addresses: dtheodor@ics.forth.gr (D. Theodoropoulos), mazumdar@dii.unisi.it (S. Mazumdar), eduard.ayguade@bsc.es (E. Ayguade), nicola.bettin@vimar.com (N. Bettin), javier.bueno@bsc.es (J. Bueno), sara.ermini@unisi.it (S. Ermini), antonio.filgueras@bsc.es (A. Filgueras), daniel.jimenez-gonzalez@bsc.es (D. Jiménez-González), aarlos.alvarezmartinez@bsc.es (C. Álvarez Martínez), xavier.martorell@bsc.es (X. Martorell), francesco.montefoschi@unisi.it (F. Montefoschi), david.oro@hertasecurity.com (D. Oro), pnevmati@ics.forth.gr (D. Pnevmatikatos), antonio.rizzo@unisi.it (A. Rizzo), pj@evidence.eu.com (P. Gai), s.garzarella@evidence.eu.com (S. Garzarella), bruno@evidence.eu.com (B. Morelli), alberto.pomella@vimar.com (A. Pomella), giorgi@dii.unisi.it (R. Giorgi).

http://dx.doi.org/10.1016/j.micpro.2017.05.018
0141-9331/



(a) AXIOM software stack (b) FPGA programming support

Fig. 1. Proposed software stack and overview of the OmpSs support for AXIOM.

cations, AXIOM provides a generic platform with its complete application development suite. Despite the existence of many FPGA-based boards, to the best of our knowledge our approach is the first that combines all the features (especially parallel programmability, connectivity and scalability). To illustrate this, we compared more than twenty boards needed for modern CPS applications, most of which coming from crowd-funding initiatives (some of which met our targets, while others did not), and present the comparisons in Table 1.

In this paper, we describe the key features of AIOM, and its progress to date. Our contributions are:

- We detail the software stack and programming model support for AXIOM based on OmpSs programming model [14].
- We illustrate in detail the low-level, inexpensive, high-speed AXIOM-Link, and the supporting OS drivers.
- We discuss the results from our design space exploration, based on the execution traces generated by OmpSs. OmpSs now supports instrumentation with Extrae [15] to generate Paraver [16] traces, for cluster and FPGA executions, for further execution analysis.
- We provide a first set of results from the project hardware prototypes.

The rest of the paper is organized as follows: in Section 2, we explain how the support for threads is provided using the AXIOM stack and the OmpSs programming model together with the profile support in Section 3; in Sections 4 and 5 we illustrate the high-speed AXIOM-Link and describe the corresponding OS drivers. In Section 6 we discuss our evaluation platform, while in Sections 7 and 8 we present our application scenarios and our experimental results. We also discuss the related works in Section 9 and finally, we conclude the paper.

2. Programming model of AXIOM

The AXIOM software stack is depicted in Fig. 1(a). In this section, we briefly describe the OmpSs programming model; the extensions planned for OmpSs to spawn tasks in the FPGA-device, and the extensions needed to support the cluster version of AXIOM.

2.1. Introduction to OmpSs programming model

The OmpSs programming model supports the execution of heterogeneous tasks written in OpenCL, CUDA, or a high-level C or C++ language that can be converted to the machine language used in GPUs or converted to the bitstream to program FPGAs. Also, the runtime supports the communications within a cluster of distributed memory machines. OmpSs can target tasks to the different nodes of the cluster. From the programmer perspective, the annotations required for the cluster support are exactly equivalent to the symmetric multiprocessing (SMP). Currently, both OpenCL and CUDA options require the programmer to provide the OpenCL or CUDA code and use the OmpSs target clauses (similar to the OpenMP target clauses) to move the data to the associated accelerator. In the AXIOM project, we are using the same technique to spawn tasks to the FPGA provided there was a compiler to generate the FPGA bitstream implementing the task, from C or C++ code or bitstream available with a known interface to access the data.

For executing tasks in the cluster version, the programmer needs to specify the task as plain C or C++ code. Execution on the OmpSs@cluster version automatically allows the runtime system to spawn tasks to remote nodes. The programming model allows parallelizing applications on the AXIOM cluster and spawn tasks on the FPGAs available on each board. Using OmpSs@cluster with FPGAs support, programmers express two levels of parallelism. The first level of parallelism targets the AXIOM-cores, i.e. the cores that are available on the AXIOM-board (e.g., the ARM-A9 cores in the case of a Xilinx Zynq SoC). Tasks at this level are spread across the AXIOM boards as if they would be executed on an SMP machine. The second level of task parallelism is expressed through the OmpSs extensions targeting the FPGAs (see below, Section 2.1.1). The OmpSs programming model is based on two main components and some additional tools. They are:

- The Mercurium compiler [17] takes the source code and understands the OmpSs directives to transform the code to run on heterogeneous platforms, including OpenCL and CUDA, accelerators. For AXIOM the compiler has been extended to generate and support FPGA-based accelerators.
- The Nanos++ runtime system [18], which is the responsible to manage and schedule parallel tasks, respecting their dependencies, transferring the data needed to/from the accelerators when needed, and the lower-level interactions.
- Additionally, OmpSs can use the Extrae tool [15] to generate execution traces that can be later visualized with the Paraver tool [16], and analyze the execution behavior.

2.1.1. OmpSs extensions for FPGAs

OmpSs was extended to support the Zynq chip with the FPGA selected in the AXIOM project. The main extension to the OmpSs programming model to provide support for these chips in the Mercurium compiler is to incorporate a new target device named `fpga`, in addition to the current `smp`, `cuda` and `opencl` devices.

Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

Table 1
Comparison of recent FPGA-based boards (AXIOM related aspects are in bold).

Boards	FPGA/CPU	RAM	Standalone	ATMEL Based Arduino	Connectivity	Programmability
LOGi FPGA	Spartan6 LX9	256MB	**No**	Pinout	SATA, Raspy/Beagle, SPI	**IDE, GUI**
MiniSpartan6+	Spartan6 LX9/25	32MB	Yes	–	I/O ports, DAC, ADC	**IDE, GUI**
Papilio DUO	Spartan6 LX9	2MB	Yes	**ATmega32U4, Mega Pinout**	I/O ports	**IDE, GUI**
MOJO	Spartan6 LX9	–	Yes	**ATmega32U4, Custom Pinout**	–	**IDE, GUI**
SmartZynq	**Zynq 7010/7020**	1GB	**No**	–	**Fast network and board2board**	Hard
Parallella	**Zynq 7010/7020 16-core Ephiphany**	1GB	Yes	–	**Gigabit ethernet, Four high speed connectors**	Standard tools
aijuboard	**Zynq 7015**	1GB	Yes	–	SATA, Gigabit ethernet	Standard tools
RED PITAYA	**Zynq 7010**	512MB	Yes	–	Gigabit ethernet, 4 fast analog inputs	Standard tools
OHO	Spartan 3E	–	Yes	–	I/O ports	Xilinx ISE only
RetroCade Synth	Spartan 3E or LX9	4 MB	**No**	–	Analog and digital inputs, MIDI, audio jacks	–
PAPILIO	Spartan 3E	8 MB	Yes	–	I/O ports	No SDK
TRIFDEV	Lattice MACHXO2-1200	–	Yes	Partial pinout, I2C	I/O ports	No SDK
owlBoard	Spartan6 LX9	–	Yes	–	I/O ports	No SDK
Alan	Spartan6 LX45	–	Yes	**ATmega32U4 and Pinout**	I/O ports	**Arduino IDE, Xilinx ISE**
4CH sig.gen.	Spartan6 LX9	–	Yes	–	4 DAC	None
Logitraxx	Spartan6 LX9	64 MB	Yes	Shield Compatible	I/O ports	No SDK
KromaLights	Spartan6 LX9 Cortex-M3	256 MB	Yes	**Arduino Due (SAM3X)**	LVDS, CAN, USART	SDK (Ardunio IDE)
CrystalBoard	Spartan6 LX9 4-core Cortex-A9	2 GB	Yes	**Atmega328, UNO Pinout**	Ethernet, WiFi	–
PSHDL board	Actel A3PN250	–	Yes	Atmel XMega32 (to program FPGA)	UART, I/O ports	Simplified VHDL
Helix-4	Altera Cyclone 4 (22k)	4 MB	Yes	Arduino UNO shield	I/O ports	Altera Quartus II IDE
ZynqBerry	Zynq 7010	128 MB	Yes	–	Ethernet	Xilinx SDK
Z-turn Board	Zynq 7010/7020	1 GB	Yes	–	CAN, Ethernet	Xilinx SDK

The `fpga` device will cause the Mercurium compiler to understand that the function annotated is to be compiled with the Xilinx Vivado HLS compiler for the FPGA in order to generate the bitstream. Other extensions may be necessary as the number of accelerators for each accelerator type.

Fig. 1(b) shows the main phases of the bitstream generation and compilation of the OmpSs code. With this extension, the compiler generates the code for the runtime system specifying the tasks that should be run in the FPGA device. The code is compiled with a back-end compiler (e.g., gcc) that will be executed in the Zynq-ARM cores. This binary code (`OmpSs.elf` in Fig. 1(b)) will call the Nanos++ runtime with FPGA execution support. This support is based on the DMA library and the FPGA-DMA driver in the system. Indeed, the tasks with `target device(fpga)` are extracted, modified and, using the Xilinx toolchain transparently to the programmer, the bitstream with the FPGA accelerators is automatically generated.

2.2. Runtime support

The runtime support has two parts: i) first part is responsible for the FPGA-based execution, ii) second part for cluster environment.

2.2.1. FPGA runtime support

The Nanos++ runtime system has also been extended, in the following ways:

- Support to spawn tasks in the FPGA device.
- Support for the target clauses related to data transfers. Data-copy clauses (`copy_in, copy_out, copy_inout`) trigger the data transfer of the data specified to/from the FPGA device. Also, `dependence` clauses will trigger data transfers to the device by default.
- Support for data transfers to/from the FPGA. The Nanos++ runtime now invokes the services of the DMA library developed to transfer data in the FPGA environment.
- Include the FPGA device in the support of the implements clause to allow several implementations of tasks to be scheduled in the available processors/devices.

In terms of FPGA support, the DMA library interface provides the means to interact with the Linux driver supporting the FPGA device. In the current prototype, when the data transferred is to the FPGA hardware, the IP kernel is initiated automatically. The computation on the data proceeds to the end, and after finishing, the results can be read back to the host from the FPGA.

The main DMA library primitives allow to get the number of IP accelerators present in the FPGA device, and the handles to operate with them. For each IP accelerator, the library allows to open input and output DMA channels to send/receive data to/from it. The library allows to allocate special memory buffers in kernel space to exchange data between the Linux kernel and the FPGA hardware. Kernel buffers are pinned to physical memory to avoid swapping them out, while a DMA transfer is in progress. Buffers can be submitted for a DMA transfer to/from the specified device. Data transfers can be monitored to determine if they are in progress, they have finished, or a transfer error has occurred. This interface is used by the Nanos++ runtime system to drive the work of the IP accelerators in the FPGA.

2.2.2. Cluster runtime support

The OmpSs@Cluster [19] approach uses a communication layer to launch tasks to remote nodes. Task descriptors and data travel on the communication layer. In our current implementation, this layer is GASNet [20], usually running on top of MPI [21] through

```
const int BS=VALUE;
#pragma omp target device(fpga, smp) copy_deps
#pragma omp task in([BS*BS]a, [BS*BS]b) \
                 inout([BS*BS]c)
void matrix_multiply(int BS,
                     float *a, float *b, float *c) {
    for (int ia = 0; ia < BS; ++ia)
        for (int ib = 0; ib < BS; ++ib) {
            float sum = 0;
            for (int id = 0; id < BS; ++id)
                sum += a[ia*BS+id] * b[id*Bs+ib];
            c[ia*BS+ib] = sum;
        }
}

...
int main( int argc, char * argv[] ){
 ...
   for (i=0; i < NB; i++) {
     for (j=0; j < NB; j++) {
       for (k=0; k < NB; k++) {
         matrix_multiply(BS,A[i][k],B[k][j],C[i][j]);
       }
     }
   }
 #pragma omp taskwait
 ...
}
```

Fig. 2. OmpSs directives on matrix multiplication.

an Ethernet link. Next step is to provide the runtime with a communication layer that can exploit the high-speed dedicated interconnection AXIOM-Link (see Fig. 1(a)) using the AXIOM network interface explained in Section 4.

2.3. Ompss coding example

Fig. 2 shows an example of matrix multiplication that has been annotated with OmpSs directives. Note that this code is independent of the execution platform (i.e., cluster, nodes with FPGAs, nodes with GPUs.), being the runtime responsible for taking care of the task execution scheduling of the tasks to the devices or nodes of the cluster, transparently to the programmer. In particular, this code shows a parallel tiled matrix multiply where each of the tiles ($BS \times BS$ sub-matrix) is a task. *A*, *B* and *C* are $NB \times NB$ matrices of pointers to $BS \times BS$ sub-matrices.

Each of those tasks has two input dependencies and an output dependence that will be managed at runtime by Nanos++. Those tasks will be able to be scheduled/fired to a `SMP` or `FPGA`, as it is annotated in the target device directive, depending on the resource availability. The `copy_deps` clause associated to the `target` directive hints the Nanos++ runtime to copy the data related to the input and output dependencies to/from the device when necessary.

3. Profiling support

The current implementation provides support to profile and trace cluster execution. At the same time, a new hardware tracing mechanism allows to profile and trace basic information from `fpga` tasks. Traces are automatically generated and translated to Paraver traces if specified at execution time. Those traces include both application and OmpSs runtime execution state information so that the programmer can analyze the parallel execution behavior to detect potential performance bottlenecks. In Section 8 some trace results are presented and discussed. Those results uncover the need for hardware profiling support.

3.1. Hardware profiling support

A new hardware support for FPGA profiling and tracing (from inside FPGA) for high-level languages has been introduced. This new feature is, to the best of our knowledge, novel for task-based parallel heterogeneous programming. The support is in the process of being integrated into the FPGA-task acceleration in OmpSs and the support is transparent to the programmer. The first profiling and tracing objective is to have input and output memory transfer, and computation information from inside the OmpSs `fpga` task execution. With this aim, the idea is to:

- Create a hardware platform that integrates hardware profiling counters that can be read from both `SMP` cores and `fpga` accelerated tasks, transparently to the programmer.
- Create hardware counters that do not affect the performance of the `fpga` tasks.
- Make the `fpga` tasks return the profiling information as part of their outputs, transparently to the programmer.
- Interpreting the profiling information in the OmpSs runtime device dependent layer, transparently to the programmer.
- Include the profiling information to the automatically generated Paraver trace.

Our implementation has used the OMPT API [22] to generate the execution traces using the Extrae instrumentation tool. The OMPT API helps to integrate profiling of different accelerators/devices and CPUs using the same API that can be supported by different instrumentation tools.

4. The AXIOM network interface

4.1. The network interconnect controller

After an initial exploration of the 32-bit Xilinx Zynq platform, the AXIOM platform is now designed around the Xilinx Zynq Ultrascale+ SoC that features a quad-core ARM A53 processor Application Processing Unit (APU) tightly coupled with FPGA fabric. AXIOM is designed to be modular at the next level, allowing the formation of more efficient processing systems through low-cost, but scalable high-speed interconnect. The interconnect will utilize the integrated gigabit-rate transceivers with relatively low-cost USB-C connectors to interconnect multiple boards. Such connectivity will allow users to build (or upgrade at a later moment) flexible and low-cost systems by cascading more AXIOM boards, without the need of costly specialized connectors and cables. AXIOM boards will feature two or four bi-directional links, so that the nodes can be connected in many different ways, such as ring and 2D-mesh/torus.

Fig. 3 illustrates the network interface (NI) architecture, originally introduced in [9], which implements remote direct memory access (RDMA) and remote write operations (raw data) as basic communication primitives visible at the application level. It consists of a queue set for RDMA and raw data messages, a set of hardware counters, control/status registers, a DMA engine, a low-level packet router, and two internal controllers for transmitting and receiving messages.

As depicted in Fig. 4, the message descriptor types handled by the NI can be divided in two main categories: *raw* messages, and *RDMA transactions* messages. Raw messages are messages for which the Network interface provides message buffers directly in the FPGA memory region. Their length is up to 128 bytes and they are used either to direct a message to a specific node (using the Node ID), or to a neighbor interface using the interface ID, in order to provide a way to implement a discovery algorithm in the Linux OS. RDMA transactions can be of three kinds: RDMA Read requests (where a node asks a copy of the memory on a remote

Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

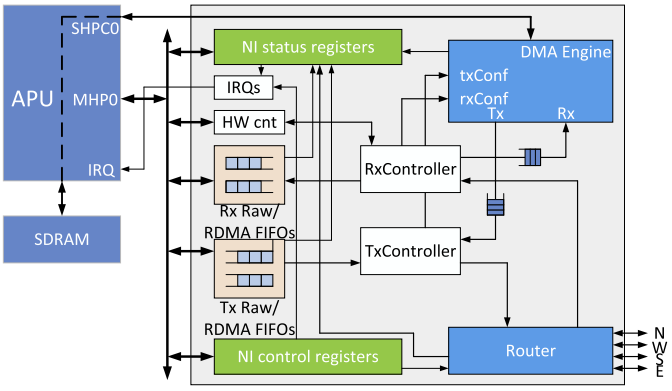


Fig. 3. The AXIOM network interface architecture.

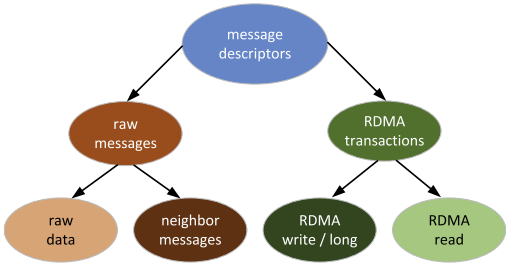


Fig. 4. Supported message descriptor types by the AXIOM interconnect.

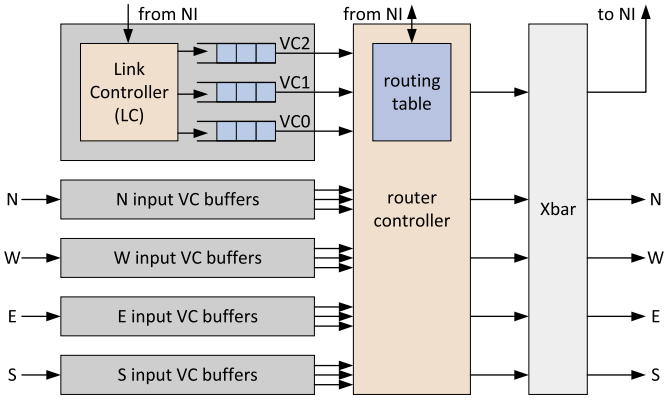


Fig. 5. The AXIOM router pipelined architecture.

node), RDMA Write requests (where a node writes on the memory of a remote node), and LONG messages (which are RDMA write requests for which the destination remote address is decided by the remote node by taking it from a set of preallocated buffers). All messages have a *port* specification, which is used in the software stack to provide separate reception queues.

To send a new message, the OS posts its descriptor to the NI queues. The local hardware counters are used to register the progress of RDMA requests (described in Section 4.2). The memory-mapped control/status registers can be used by the OS to configure notification parameters (e.g., acks at OS-level upon successful packet transmission and IRQs), and monitor the progress of RDMA requests, respectively. The DMA engine is used for loading and storing data from/to the local SDRAM. The local APU communicates with its local NI via the Master High-Performance Port 0 (MHP0) and Slave High-Performance Coherent Port 0 (SHPC0) interfaces. MHP0 is used to access the control/status registers, while SHPC0 allows fast and coherent data storage to the local external memory.

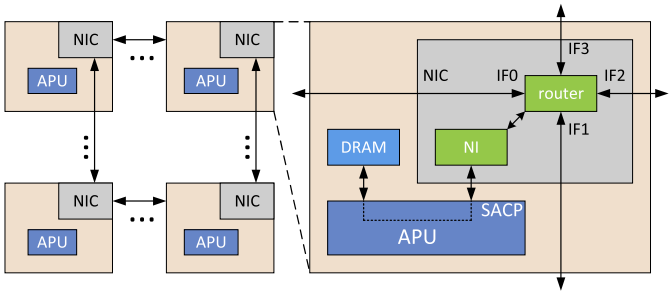


Fig. 6. AXIOM boards interconnected in 2D-mesh.

The router module shown in Fig. 5 implements the routing and network discovery processes. The AXIOM routing algorithm will feature store-and-forward packet transmission with virtual circuits (VCs), and the network discovery process will be initiated at boot time by the master node of network. After the process completion, every node will have its id, and local routing table, based on which all packets will be forwarded to output links.

The core router components can be outlined as: i) input buffering, ii) control, and iii) crossbar and link traversal. The input buffering module consists of four link controllers (LC), where each link employs queues to implement three VCs to store different priority packets. The router uses a Xon/Xoff strategy for notifying adjacent nodes on VC input buffer availability. If a VC queue reaches a predefined threshold, the router instantly transmits a Xoff packet to the link's adjacent node to block further packets transmission. Similarly, when the VC fullness drops below a certain level, the router instantly transmits a Xon packet to the link's adjacent node to resume packets transmission via this particular link.

The route calculation (RC) finds the required output interface for a packet, based on the routing table and destination node, starting from the highest VC. If the VC number of the output link is enabled, then the packet is forwarded to the corresponding VC allocation (VCA). For each input link, the VCA always attempts to serve the VC with the highest priority, except if its destination node input VC buffer is blocked. In that case, it falls to the next lower input VC.

During the switch allocation process, the packets from each buffer request a Xbar output. The switch allocation pairs the Xbar inputs to the Xbar outputs as efficiently as possible, trying not to leave an output link idle. If more than one packets request the same output link, the grant policy decides according to:

- Priority (Xon/Xoff >VC2 >VC1 >VC0).
- If packets are of the same priority (e.g., both VC2), it chooses one (in a round-robin based fashion) to grant an output port, while at the same time looks for available packets of lower priority (VC1 or VC0) on the same input link that requires a different port.
- Repeat until all packets are served.

The use of VCs with priorities ensures that we avoid protocol deadlocks in the network. As we use low priorities for requests, medium priorities for responses and a top priority for acknowledgments, there is no possibility for high-priority packets to clog the network as their priority will be less or equal than the requests that were accepted by the network. Thus in the case of a high network congestion, acknowledgments will exit the network, then responses will be sent, and then more requests will be accepted. The crossbar module is responsible for forwarding all available packets to their output links. All packets then traverse via the physical link to the neighbor node and are stored to the corresponding VC input queue. Finally, Fig. 6 shows how AXIOM boards can be interconnected in 2D mesh.

Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

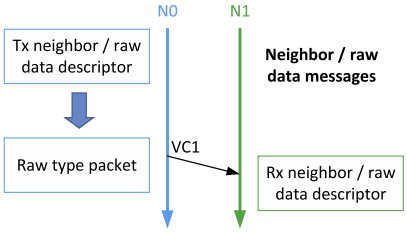


Fig. 7. Neighbor/raw data messages flow.

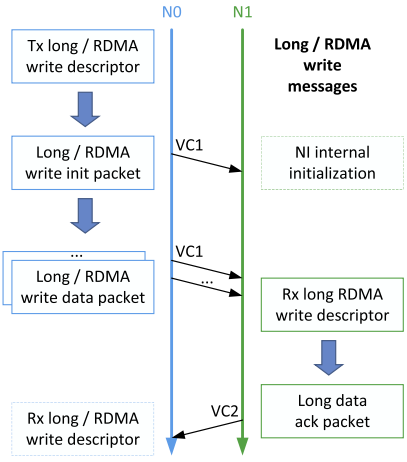


Fig. 8. Long/RMDA write messages flow.

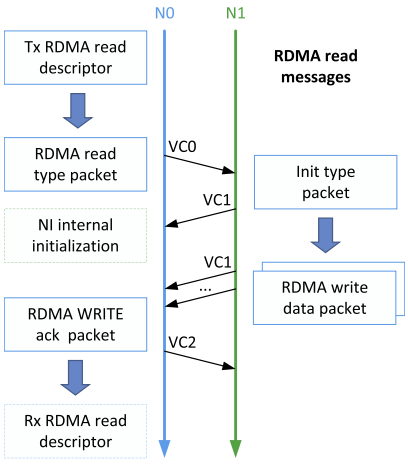


Fig. 9. RMDA read messages flow.

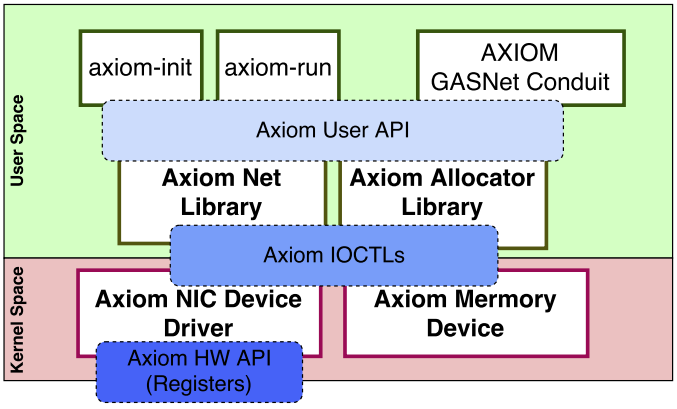


Fig. 10. The AXIOM network stack architecture.

4.2. Packet flow

Fig. 7 illustrates the flow of transmitting raw/neighbor messages. The OS hosted on node N0 posts a Tx neighbor/raw descriptor to the NIC queues for transmission. The NIC controller pops the descriptor, creates a raw packet and transmits it via VC1 to N1. When the N1 NIC receives the packet, it posts an Rx neighbor/raw descriptor that the N1 OS reads and extracts the payload.

Fig. 8 depicts the procedure of transmitting long/RDMA write messages between node N0 and N1. The N0 OS posts a Tx long/RDMA write descriptor to the NIC queue. The NI internal controller parses the descriptor and transmits to N1 via VC1 an init packet that designates the data payload size, followed by long/RDMA write packets, each "carrying" a subset of the requested data. On the N1 side, when the init packet is received, the local NI associates a local hardware counter with the corresponding message, and initializes its value to the number of expected long/RDMA write packets. As soon as all packets are received, the NI posts an Rx long/RDMA write descriptor that the N1 OS can parse it to retrieve the requested data. Moreover, the N1 NIC sends an ack packet via VC2 to N0 with the total number of bytes received. When the N0 NIC receives it, if already configured by the OS, it can post an Rx descriptor to signal the N0 OS that the long/RDMA write is successfully transmitted.

Finally, Fig. 9 shows the transmission procedure of RDMA read messages between node N0 and N1. The N0 OS posts a Tx RDMA read descriptor to the NIC queue. The NI controller parses the descriptor and transmits via VC0 an RDMA read request packet. On the N1 side, when the RDMA read packet is received, the local NI essentially follows the RDMA write procedure described above, in order to transmit all requested data to N0. N0 associates a local hardware counter with the corresponding message, and initializes its value to the number of expected long/RDMA write packets. Again, when all packets are received, the N0 NI transmits an ack packet to N1 with the number of bytes that were received, and also posts an Rx RDMA read descriptor to signal the local OS that all requested data have arrived to the designated address.

5. The AXIOM network drivers

The AXIOM network stack is a software stack developed on top of Linux which is meant to provide an efficient access to the AXIOM NIC features, including Remote DMA transfers.

The network stack (see Fig. 10) is composed by a Linux kernel driver that provides a proper interface for the user libraries, exposing high-level constructs that are then mapped into the NI registers. An additional kernel driver also takes care of the memory allocation (see Section 5.2). In the user space, a set of libraries and daemons (see Section 5.3) are used to provide user space services to the AXIOM applications.

5.1. Network interface kernel drivers

The architecture of the kernel drivers handling the network is depicted in Fig. 11. The main components of the stack are the following:

- A set of software queues (one per port) for the small messages;
- A set of software queues (one per port) for the descriptors of the long messages;
- A RDMA queue to store the descriptors of the RDMA requests;
- A pool of descriptors that are pre-allocated by the driver to be used to automatically allocate long messages upon their arrivals;

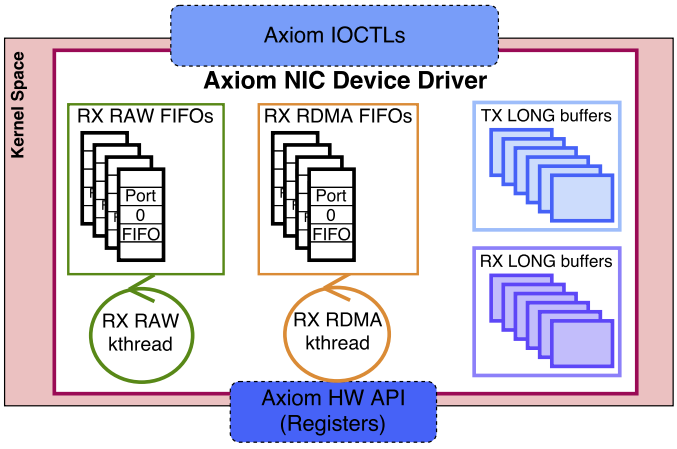


Fig. 11. The AXIOM network stack architecture.

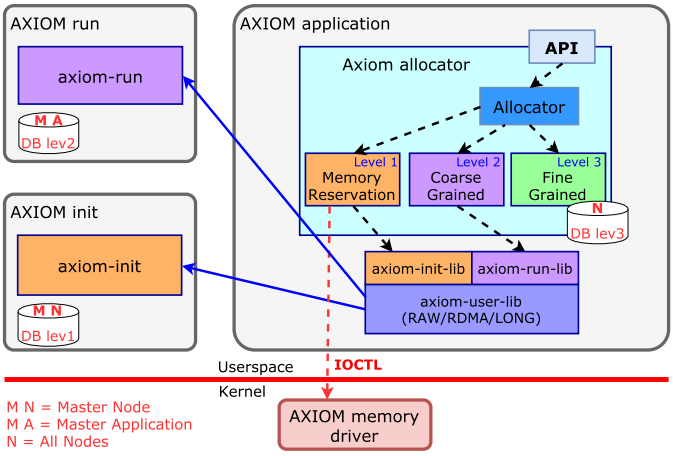


Fig. 12. The three levels of the AXIOM Allocator.

- A set of kernel threads that are responsible for polling the incoming message queues, demultiplexing their content into local kernel-level buffers, and for filling the long message descriptor FIFOs.

All these components exports custom IOCTL commands to userspace applications (see Section 5.3).

An additional kernel driver is dedicated to the memory management. The main idea of the driver is that each board dedicates a contiguous physical memory range to RDMA transactions. That memory range is handled by the AXIOM memory driver, which is responsible of assigning subsets of that memory range to the various user processes. These per-process memory assignments are used by the AXIOM memory allocator to provide dedicated and shared memory across the AXIOM cluster.

5.2. Memory allocator

The AXIOM memory allocator is responsible for the memory subsystem used by the AXIOM drivers and by the AXIOM applications.

The starting point is the physical memory available to all nodes. We make the hypothesis that each node has a physical RAM memory mapped at similar addresses (this is true in the case of AXIOM, where the cluster is composed by homogeneous nodes). The physical memory available is at least partly RDMA-addressable.

The main idea behind the AXIOM allocator is to handle a part of the memory of each node in the AXIOM cluster in a way compatible with the RDMA support of the AXIOM NIC described in Section 4.1. This is solved on each node by reserving a dedicated range of contiguous physical memory; that reserved memory is outside the memory range directly managed by the Linux kernel, and is then managed by the AXIOM allocator which will be responsible to map memory regions to the various processes composing an AXIOM application.

The kind of memory that can be allocated by the AXIOM allocator is either a *private* memory or a *shared* memory. Allocating *private* memory guarantees unique address ranges only on the *node* requesting it (that is, two nodes may end up allocating private memories at the same virtual address). Allocating *shared* memory guarantees that the range of memory allocated is unique among *all the AXIOM cluster*. Note that the allocator provides guarantees on the uniqueness of the address in the cluster, but not on its coherency or synchronization, which is guaranteed by the higher software layers based on DF-threads or OmpSs/GASNet.

The AXIOM allocator is internally composed by three levels (see Fig. 12):

Level 1 is responsible for reserving regions of memory at *cluster* level. The idea is that this reservation is only inquired at start/end of an application to reserve the maximum (shared or private) memory used by the application. This level is implemented partly inside the device driver (to enforce memory mapping and correct memory addressing), and in the `axiom-init` application (on the master node of the cluster, see later).

Level 2 is responsible to allocate macro-blocks of shared memory to specific nodes. In other words, this coarse-grained allocator is responsible for guaranteeing that the allocation of shared memory will return unique addresses at cluster level. This level is implemented inside the `axiom-run` application (on the application master node, see later).

Level 3 is finally responsible of each single allocation of private/shared memory. Various allocators can be supported at this stage, such as LMM [23,24] and in the future TLSF [25]. These fine grained allocators will work locally on each node providing quick and efficient allocation of memory that has been reserved by Level 1 and 2.

5.3. User space applications and service libraries

The AXIOM Network drivers have been developed together with a set of applications that complement the driver functionality.

The first application we will describe is the `axiom-init` Linux daemon. The idea is that `axiom-init` includes in user space some of the services that normal networks (like TCP/IP) include in their kernel layers. This approach has the advantage to limit the size of the AXIOM drivers, leaving everything which is configuration dependent in user space (thus allowing easier changes). `axiom-init` is responsible for the following services:

- It handles the initialization part of the cluster. In particular, it is responsible for running the AXIOM discovery algorithm (used to discover the topology of the network and set up the IDs of the nodes), for computing the node routing tables and setting them on each node of the cluster.
- It is responsible for a set of diagnostic protocols that are handled with a set of separate applications like `axiom-ping` (note that in TCP/IP implementations these services are typically implemented in the kernel driver; in AXIOM, `axiom-init` provides the same kind of support but in user space).
- It handles the cluster level synchronization needed by the Level 1 of the AXIOM allocator. In particular, it stores the data structures that keep track of the various allocations in the first node of the cluster (named *master node* of the cluster).

Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

- It provides a set of services for starting applications, like providing the application ID, as well as a service to spawn processes in the cluster.

Another set of applications is then provided to implement a simple messaging and diagnostic interface. In this list, we include:

- `axiom-info`, which is used to provide information on the node ID, on the routing table on the local node, and on the set of interfaces available on the node.
- `axiom-traceroute`, `axiom-netperf`, `axiom-ping`, which are used to provide services similar to their Unix counterparts.
- `axiom-send`, `axiom-recv`, `axiom-rdma`, which are used to send and receive long, raw messages, and to trigger test RDMA operations.

Finally, another fundamental application in the AXIOM stack is `axiom-run`. `axiom-run` is used to provide a set of services to AXIOM applications:

- An AXIOM application running on the cluster is composed by a single executable which is run once for each node of the cluster. `axiom-run` provides the AXIOM application startup procedure, allowing the possibility to spawn a process on a subset of the nodes of the cluster. In order to do that, it uses the spawn service provided by `axiom-init` to start a process on a single node.
- It provides the support for application termination. In particular, in case one of the processes spawned on a node terminates, it is responsible to terminate all the other processes on the nodes of the cluster.
- It provides a standard output redirection service for the spawned processes. In other words, the standard output of the spawned process is captured by *slave* executions of `axiom-run` on each node, and redirected to the *master* application node (that is the node from where the user started the application initially).
- It provides other additional services to the AXIOM applications such as a synchronization *barrier* for all nodes running the application, and the Level 2 of the AXIOM allocator (which requires synchronization only at application level). These additional services are located in the first node used by the application, which is named *application master node*.

`axiom-run` and `axiom-init` are provided together with their respective user libraries. These user level libraries are used to let third party applications interact with them in a simpler way.

6. AXIOM Evaluation Platform (AEP)

Design space exploration (DSE) and its automation is an important part of our current performance evaluation and power estimation methodologies [26–28]. The proposed method in AXIOM requires first exploring and modeling parts on the simulator and then, once the DSE is completed, implementing them on the FPGA-based prototypes. This has the considerable advantage of allowing immediately to develop the software stack early. AEP is made of two important tools: the HP-Labs COTSon simulator [29] and the Xilinx Zynq based platform. Given the goals of this project, we also needed a more flexible platform for the DSE. The simulation platform is used to understand better bottlenecks (e.g., the congestion on a bus, cache size), which are not trivial to track on the FPGA prototyping platform. COTSon also includes an interface to the HP McPAT tool [30] for estimating the power consumption. Table 2 presents some advantages of using COTSon for our purpose.

COTSon uses the so-called "functional-directed" approach. The simulator permitted us to execute the full-system simulation. The "mediator" of COTSon represents the model of a switch, and our aim is to modify it to model the behavior of our custom interconnects. The motivation for multiple interconnects derives from the AXIOM project design that aims to separate the traffic for building a multi-board system and the traffic for the internet related connection. With the COTSon mediator, we can model both cases. The SimNow is the virtual machine (VM), which models all details of a computer. AMD is also providing a separate SDK to model any particular board that has to be plugged-in (such as a network card or a GPU).

Table 2
Comparison of COTSon with other DSE environments.

Features	Sniper	Graphite	Gem5	MARSx86	COTSon
Timing directed	No	No	Yes	No	No
Functional directed	Yes	Yes	No	Yes	Yes
User level	Yes	Yes	Yes	No	No
Full system simulation	No	No	Yes	Yes	Yes
Parallel (In node)	Yes	Yes	No	No	No
Parallel (Multi-node)	No	No	No	No	Yes
Shared cache	Yes	No	Yes	Yes	Yes

6.1. Thread support

Synchronization and distribution of data can be managed efficiently by reorganizing the execution in such a way that the threads follow more closely the data flow of the program (such as with DF-Threads [31]). DF-Threads can be efficiently implemented by a distributed hardware thread scheduler which support fault tolerance at the hardware level and efficient fine grain dataflow thread distribution [32]. To reduce the thread management overhead, the scheduling needs to be accelerated in hardware, by mapping its structure into the FPGA. A DF-Thread is defined as a function that expects no parameters and returns no parameters. The body of this function can refer to data which reside at the memory location for which it has got the pointer. The DF-Thread API's are summarized below [33]:

- `void *DF_TSCHEDULE(bool cnd, void *ip, uint64_t sc)`: Allocates the resources (a DF-frame of size `sc` words and a corresponding entry in the distributed thread scheduler or DTS) for a new DF-Thread and it returns a frame pointer `fp`. The `ip` is the instruction pointer of DF-Thread. The allocated DF-Thread is not executed until its `sc` reaches 0 and together also satisfy the boolean condition `cnd`.
- `void DF_DESTROY()`: To release allocated resources held by current DF-Thread.
- `uint64_t DF_TREAD(uint64_t offset)`: Loads the data indexed by `offset` from the current thread of DF-frame.
- `void DF_TWRITE(uint64_t val, void *fp,uint64_t off)`: The data `val` is stored into the DF-frame pointed to by `fp` at the specified offset `off`.
- `void *DF_TALLOC(uint64_t size, uint_8 type)`: Allocates a block of memory of `size` words and returns the pointer (or null) while `type` specifies the special purpose memory type.
- `void DF_TFREE(void *p)`: Frees memory pointed to by `p`.

7. Application scenarios of AXIOM

Smart video surveillance and smart home applications are now a hot topic in CPS and we have customized these two scenarios for our AXIOM platform.

A. Smart Video Surveillance (SVS)

For SVS case study, we selected an automated smart marketing scenario involving real-time face detection in crowds while per-

Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

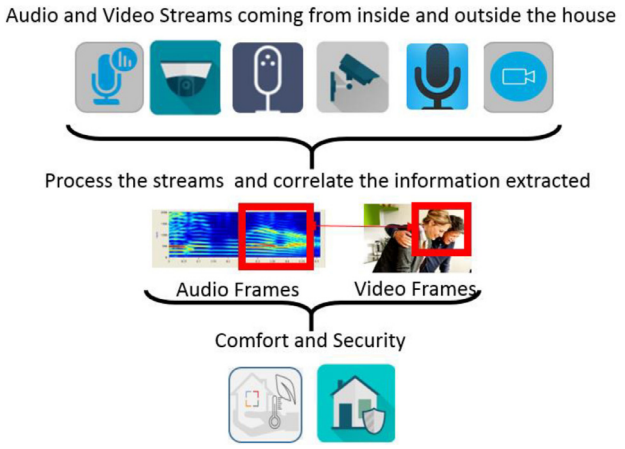


Fig. 13. The AXIOM smart home living (SHL) scenario.

forming demographics estimation (e.g., age, gender and ethnicity). The SVS scenario will employ state-of-the-art cognitive computer vision techniques based on models built from a boosted cascade of classifiers combined with deep convolutional neural networks. A low-power high-performance inference engine for such models will be implemented in the reconfigurable logic of the SoC using the OmpSs programming model. Since this scenario will analyze high-definition (HD) video feeds, other computational challenges related to video processing must also be addressed. HD video stream decoding (i.e., format parsing, codec implementation, demuxing and color space conversion) will be performed by relying on a heterogeneous computing approach combining single instruction, multiple data (SIMD) instructions with on-die logic blocks.

B. Smart Home Living (SHL)

Regarding the SHL case study, a solution to enhance the security level of the house and to increase the comfort of the smart home has been implemented. The solution developed consists on a system that is able to analyze multimedia streams captured in specific points inside and outside the house. Fig. 13 shows an overview of SHL scenario. The system receives the multimedia streams from the networks, splits and decodes the audio and the video streams and analyses the raw data using machine learning algorithms to extract information from the two media components. The information extracted from the media data are correlated to define the feedback that will be sent to the user of the house. The main goal of this project is to achieve a high level of automation and allowing a natural interaction between the user and the house. To achieve this goal is required that the time to analyze the data and to produce the feedback should not interrupt the user's actions flow and this represents a strict timing constraint for our system. To take advantage of the heterogeneity and the cluster architecture of the AXIOM system the OmpSs directives will be introduced in the code of the SHL application. Different solutions will be explored for the purpose of defining the tasks that can be concurrently executed and defining the granularity of these to satisfy the requirements of the SHL application. OmpSs@FPGA directives will be used to efficiently synthesize the most time consuming sections of the algorithms on FPGA resources and OmpSs@Cluster will be used to split the execution of the application in different nodes of the cluster to gain the timing constraints and at the same time to minimize the hardware resources and energy consumption.

8. Evaluations and results

In this section, we present some preliminary results for some software and hardware prototypes the AXIOM project is designing and implementing.

Table 3
OmpSs experimental results: we use as many worker threads as number of cores.

Machine	w/o NEON		w/ NEON		Speed-up
	Time (s)	GFLOPS	Time (s)	GFLOPS	
UDOO: 1 core (1 node)	7.6	0.28	2.90	0.74	2.6×
UDOO: 4 cores (1 node)	1.9	1.13	0.96	2.20	7.9×
UDOO: 8 cores (2 nodes)	1.3	1.61	0.75	2.84	10.1×
Zynq 706 board (FPGA)	Times = 0.5s		GFLOPS=4.06		15.3

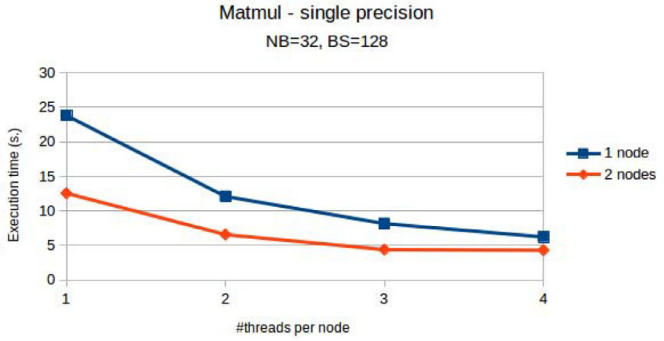


Fig. 14. Execution time for the matrix multiply of size 2048 × 2048 and blocks of 128 × 128 using NEON SIMD instructions.

8.1. Ompss timing results

Table 3 shows the execution time and GFLOPS of the matrix multiplication of Fig. 2 for different execution environments. Those environments are: i) one core of the UDOO × 86 cluster [34,35], using/not using NEON SIMD instructions, ii) all cores (four) of the same node of the UDOO cluster, with or without NEON instructions, iii) all cores of the two-node UDOO cluster, with and without NEON instructions and iv) a Zynq ZC706-SoC using the FPGA to accelerate the matrix multiply tiles. All the results are for a tiled matrix multiply with BS=128 and 1024x1024 matrices. The number of worker threads to perform the computation is the same as the number of cores used. Speedup results are obtained comparing each environment result, with NEON instructions or FPGA, to the UDOO 1 core environment without NEON instructions. On one hand, the use of NEON SIMD instructions significantly improves the application performance, and in general, it seems that there is good scalability inside one node. Nevertheless, the Zynq ZC706 board result, using FPGA accelerators for the matrix multiply, shows a much better performance than the UDOO cluster. It can be stated that the AXIOM platform will outperform the UDOO cluster (using NEON instructions) by 1.5×, if only the FPGA is used in each node.

Fig. 14 shows the execution time of a matrix multiply with BS=128 and 2048x2048 size for 1 and 2 UDOO nodes and 1 to 4 worker threads per node for the case of using NEON SIMD instructions. With only one node and 4 threads, the OmpSs matrix multiply already achieves a speed-up of 3.8× compared to the case of 1 node and 1 thread; showing that OmpSs@cluster scales pretty well inside one node. However, it seems that there are some overheads that reduce the scalability when using the two nodes of the cluster. One possible reason is the fact of using one helper thread to do the communication management, only necessary when having more than one node in the cluster. That provokes oversubscription when using 4 worker threads, plus the communication helper thread, in the 2 nodes case (with only 4 cores per node). Although for large systems this is not an issue, improvements on this may benefit applications using the full system resources. Furthermore, the connection done by Ethernet may also introduce synchronization and communication overheads. Therefore, the use of the high-

Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

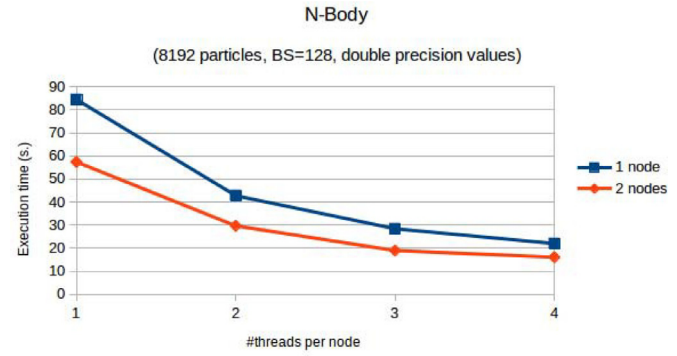


Fig. 15. Execution time of the OmpSs N-body using 1 and 2 UDOO nodes, with up to 4 threads per node.

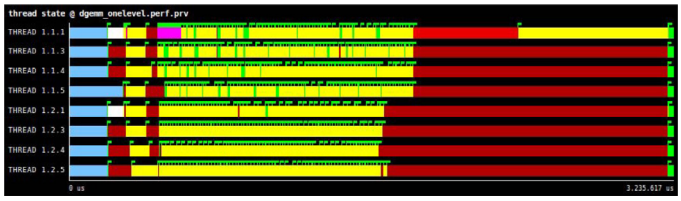


Fig. 16. Paraver trace of the OmpSs M× M using 2 nodes UDOO x86, with 4 threads per node.

speed dedicated interconnection AXIOM-Link should help to reduce those overheads and improve the scalability of the OmpSs applications.

Fig. 15 shows the execution of the OmpSs N-Body with `target device(smp)` for 1 and 2 nodes and blocksize $BS = 128$, and from 1 to 4 threads per node. The number of iterations done in the N-Body is 100 and the number of particles is $8K$. In this case, the block-size determines which amount of particles forces or updated particle positions should be computed by one task. Results show, as for the matrix multiply, that OmpSs@Cluster scales pretty well inside one node meanwhile it seems that the scabilitiy when using the two nodes is not ideal. The reason seems to be the same as before: the overhead of communication and the oversubcription due to the communication helper thread, and then, again, the use of the interconnection AXIOM-Link should help.

8.2. Profiling and tracing results

In this sub-section, profiling and tracing results are presented. Cluster profiling results have been obtained using a cluster of UDOO × 86's; meanwhile, one node traces with `fpga` task executions are on a Zynq 706 board.

8.2.1. Cluster profiling

Fig. 16 shows the execution of the OmpSs matrix multiply (BS=128) in Fig. 2 with `target device(smp)`. The cluster has two nodes with four threads per node, each of them executing `smp` tasks. The Paraver trace has as many horizontal lines as threads running OmpSs tasks. The different colors mean different thread states along the execution time of the application. Therefore, there are eight horizontal lines (one per thread). `Green` flags indicate trace events (e.g., start/end a task). Main area colors in the trace have the following meaning: `pink` areas correspond to the task creation on the master thread (top), `yellow` areas correspond to `smp` tasks running in the SMP, `light red` in the master thread (first horizontal line) corresponds to a global task synchronization, and `dark red` corresponds to idle state where those threads are doing nothing. The trace shows that tasks have been evenly distributed among the two UDOO's nodes, achieving a promising performance result. In this Paraver trace, the dependences between tasks have not been shown for clarity purposes.

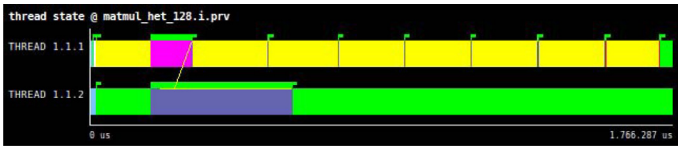


Fig. 17. Paraver trace of the OmpSs MxM using 1 SMP (top) and 1 helper thread (bottom) for two FPGA accelerators.

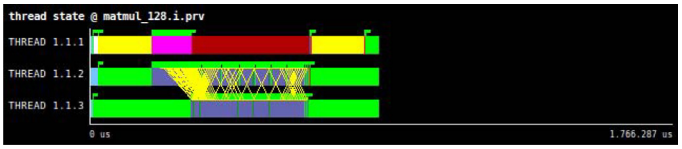


Fig. 18. Paraver trace of the OmpSs M× M using a master thread (top) to submit tasks to two FPGA accelerators (the two on the bottom).

8.2.2. One node profiling

In this case, and for the purpose of presenting an execution trace that helps to detect a performance bottleneck, we have selected a sub-optimal hardware/software co-design of the parameters and task target devices of the tasks of an OmpSs application in one node and accelerating tasks in the FPGA. Therefore, Fig. 17 shows a Paraver trace of the parallel execution of the OmpSs matrix multiply (BS=128) in Fig. 2, running in a Zynq machine and using one thread for `smp` task executions and one thread for `fpga` task submissions to two accelerators. In this Paraver trace, there is one thread (top) running tasks in the SMP and one thread (helper thread) submitting tasks to two MxM accelerators in the FPGA (bottom). `Green` flags indicate trace events (e.g., start/end a task) and `Yellow` lines between events/states indicate task dependences. Main color areas in the trace have the following meaning: `pink` areas correspond to the task creation on the master thread (top), `yellow` areas correspond to `smp` tasks running in the SMP and `purple` areas are for the submission of `fpga` tasks to one of the FPGA accelerators. `Light green` in the helper thread corresponds to thread waiting for more tasks to be submitted to the FPGA.

On one hand, this execution trace shows significant load imbalance between the two threads. The reason is the decision of executing tasks in SMP when the FPGA, at the same task granularity, is much faster than the SMP. The programmer could decide either to specify only `fpga` tasks and/or change the task granularity at SMP. In fact, for the same task scheduling policy, when the programmer decides to specify only `target device(fpga)` for the MxM task the performance is much better. Fig. 18 shows an execution trace for this scenarios at the same scale than the previous execution trace; achieving a speed-up of more than 2× (considering the matrix multiply part of the execution trace).

On the other hand, those traces do not give much information about the memory transfer (DMA) from/to Host/FPGA, possible overlapping of memory transfers and FPGA acceleration, and FPGA computation time in the two accelerators. For this reason, it is important to have hardware profiling support to provide useful FPGA profiling information to the programmer, from inside the FPGA. Fig. 19 shows the zoom in of an execution trace where it is shown the information of the DMA tranfers (input - `DMA_in` and output - `DMA_out`) and FPGA acceleration computation when using two FPGA accelerators. This helps to deepen the analysis of the performance application. For instance, it is possible to see that `DMA_in`, `DMA_out` and the FPGA accelerations of the two different accelerators may be overlapped during the execution, that the `DMA_in`

Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

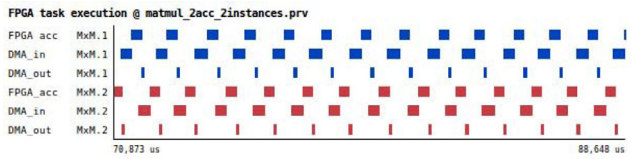


Fig. 19. Partial view of a paraver trace of the OmpSs M× M where it is shown the DMA transfers (`DMA_in` and `DMA_out`) and computation time (FPGA acc) for two accelerators.

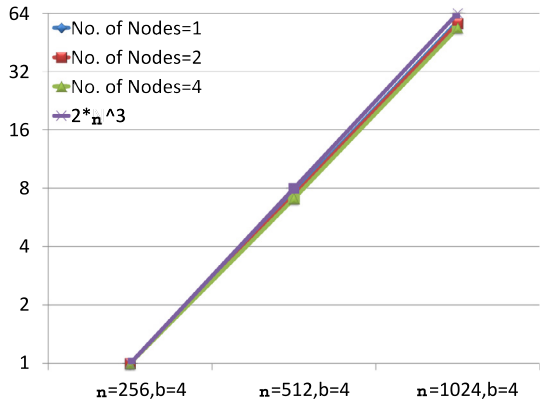


Fig. 20. Instruction count normalized to the matrix size 256 (n=256) and b is the block-size.

execution time is around 3 times the `DMA_out`, and very similar to the FPGA acceleration computation. With that, the programmer and programming model developer can detect bottlenecks on the applications or the task execution model.

8.3. DF-Threads initial results

We are reporting in this sub-section our experimental results when AXIOM platform consists of 1, 2 or 4 nodes. In this case, the execution model is based on the DF-Threads and the methodology illustrated in Section 6. For simplicity, we use a well-known benchmark which is the blocked matrix multiplication (see Fig. 2). The parallelization is based on the ratio between the matrix size `n` and the block size `b` (i.e., the expected number of DF-Threads is `n/b`). In our experiment, we consider three matrix sizes: `n=256, 512, 1024` while the block size is fixed to `b=4` and we report the results in Fig. 20 and in Fig. 21. In such cases, the number of DF-Thread is respectively `64, 128, 256`. The interesting result is related to the total number of instructions. As we can see from Fig. 20, for each matrix size the instruction count has almost the same value once we vary the node size from 1 to 4 (three superposing lines). The reason for that is due to the small overhead to manage DF-Threads across nodes. Moreover, the number of instructions follow the theoretical increase (i.e., the number of instructions increases as $O(n^3)$) in the case of a classical block-matrix multiplication closely. We normalized the total number of instructions for each curve to the case of matrix size `n=256` to compare the three experimental cases and the theoretical $O(n^3)$ line in Fig. 20.

As we can see from Fig. 21, the scalability improves significantly when we have a larger number of threads. In the case of `n=1024, b=4` the speedup is almost ideal (for four nodes the speedup is almost 4). We do not report here the effect of different block sizes, but for smaller block sizes we typically achieve better scalability [36]. What has to be stressed here is the possibility to scale performance across nodes that have separate address spaces.

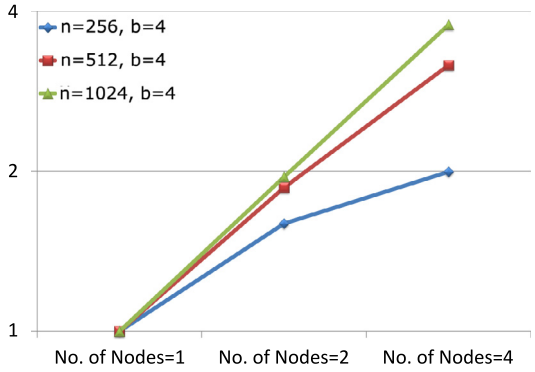


Fig. 21. Speed-up of user cycles count normalized to the matrix size 256 (n=256) and b is the block-size.

9. Related works

In last few years CPS domain has gained tremendous importance, and as a consequence lot of works been done by the academia as well as industry. We refer readers to the general survey on CPS [37] for more information. There are many similar/relevant completed (such as CONTREX [10], MultiPARTES [13], ASAM [38], SCUBA [39]) and on-going European projects (such as DREAMS [11], EMC^2 [12]) which are focused on many different aspects (mostly focused on the mixed-criticality application domain) of CPS. We are discussing few of them which are more relevant to our works.

CONTREX project mainly focused on developing energy efficient and low-cost hardware design for embedded mixed-criticality system based applications (such as automotive, aeronautics and telecommunications). The primary aim of CONTREX is to enable energy and cost efficiency through the analysis and optimisation of real-time, power, and temperature based on different criticality levels demands. The main objective of MultiPARTES is to provide an execution environment and tools to support the development of mixed-criticality applications. It offers multicore platform based virtualisation layer over partitioned embedded platforms to separate the execution environment in multicore systems. Completed EU projects such as SCUBA also developed a CPS architecture for self-organizing, cooperative and robust building automation systems (BAS) [39]. Similarly, automatic architecture synthesis and application mapping (ASAM) targeted a uniform process for heterogeneous multicore embedded systems based on application specific instruction-set processors. It aims at defining a new design environment by providing unified design methodology and set of tools to allow rapid exploration of algorithms, architecture design spaces, and also system-level synthesis.

Ongoing EU project such as DREAMS focuses on the cross domain architecture based on open-source (XtratuM) virtualization and design tools for supporting execution of mixed critical applications on networked multicore chips; *EMC*^{2} project provides a flexible MPSoC architecture which can be tailored by middleware for executing real-time and mixed-criticality applications. These mentioned projects are highly focused on (mixed) critical applications and evaluate their platforms mostly on avionics, wind power based application domains. However, AXIOM provides 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.

10. Conclusions

In this paper we presented the AXIOM platform, which provides an integrated approach including a heterogeneous SoC (cur-

rently with an FPGA) board, a new high-performance connection link to form clusters of processing nodes, and the task-based programming model, that can support single and multiple-node heterogeneous parallel execution, transparently to the programmer.

To evaluate the AXIOM platform, we ran two well-established micro-benchmarks, namely the matrix multiplication and the N-body simulation on the project software and hardware platforms. Results show that performance scales well with respect to the number of deployed processing nodes, while keeping the development effort low for application programmers.

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 through TIN2015-65316-P project, and by the Generalitat de Catalunya (contracts 2014-SGR-1051 and 2014-SGR-1272). We also thank the Xilinx University Program for its hardware and software donations.

References

[1] Cyber Physical Systems Public Working Group and others, Framework for cyber-physical systems, Preliminary Discussion Draft, Release 0.8. 2015.
[2] J. Sztipanovits, S. Ying, Strategic R&D Opportunities for 21st Century Cyber–Physical Systems, Technical Report for Steering Committee for Foundation in Innovation for Cyber-Physical Systems: Chicago, IL, USA, 13 March, 2012 Technical report.
[3] E. Geisberger, M. Broy, Living in a networked world: Integrated research agenda cyber-Physical systems (agendaCPS), Herbert Utz Verlag, 2015.
[4] E.A. Lee, Cyber physical systems: design challenges, in: Object Oriented Real–Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on, IEEE, 2008, pp. 363–369.
[5] R. Baheti, H. Gill, Cyber-physical systems, Impact Control Technol. 12 (2011) 161–166.
[6] T. Sanislav, L. Miclea, Cyber-physical systems-concept, challenges and research areas, J. Control Eng. Appl. Inform. 14 (2) (2012) 28–33.
[7] President's Council of Advisors on Science and Technology (U.S.) and Marburger, J.H. and Kvamme, E.F. and United States. Executive Office of the President, Leadership under challenge: Information technology R&D in a competitive world. An assessment of the federal networking and information technology R&D program, Technical Report, DTIC Document. 2007.
[8] D. Theodoropoulos, D. Pnevmatikatos, C. Alvarez, E. Ayguade, J. Bueno, A. Filgueras, D. Jimenez-Gonzalez, X. Martorell, N. Navarro, C. Segura, C. Fernandez, C. Fernandez, J.R. Saeta, P. Gai, A. Rizzo, R. Giorgi, The AXIOM project (agile, extensible, fast i/o module), in: Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on, IEEE, 2015, pp. 262–269.
[9] C. Alvarez, E. Ayguade, J. Bueno, A. Filgueras, D. Jimenez-Gonzalez, X. Martorell, N. Navarro, D. Theodoropoulos, D.N. Pnevmatikatos, D. Catani, C. Scordino, P. Gai, C. Segura, C. Fernandez, D. Oro, J.R. Saeta, P. Passera, A. Pomella, A. Rizzo, R. Giorgi, The AXIOM software layers, in: Digital System Design (DSD), 2015 Euromicro Conference on, IEEE, 2015, pp. 117–124.
[10] K. Grüttner, R. Görgena, S. Schreinera, F. Herrerab, P.P. nilb, J. Medinab, E. Villarb, G. Palermoc, W. Fornaciaric, C. Brandolesec, D. Gadiolic, E. Vitalic, D. Zonic, S. Bocchiod, L. Cevae, P. Azzonif, M. Poncinog, S. Vincog, E. Maciig, S. Cusenzah, J. Favaroh, R. Valenciai, I. Sanderj, K. Rosvallj, N. Khalilzadj, D. Quagliak, CONTREX: design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties, in: 2016 Euromicro Conference on Digital System Design (DSD), 2016, pp. 286–293.
[11] R. Obermaisser, Z. Owda, M. Abuteir, End-to-end real-time communication in mixed-criticality systems based on networked multicore chips, in: Euromicro Conference on Digital Systems Design, 2015, pp. 293–302.
[12] W. Weber, A. Hoess, F. Oppenheimer, Emc2 a platform project on embedded microcontrollers in applications of mobility, industry and the internet of things, in: Digital System Design (DSD), 2015 Euromicro Conference on, IEEE, 2015, pp. 125–130.
[13] S. Trujillo, A. Crespo, A. Alonso, J. Pérez, Multipartes: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems, Microprocess. Microsyst. 38 (8) (2014) 921–932.
[14] A. Duran, E. Ayguadé, R.M. Badia, J. Labarta, X.M. Luis Martinell, J. Planas, OMPSS: a proposal for programming heterogeneous multi-core architectures, Parallel Process. Lett. 21 (02) (2011) 173–193.
[15] Barcelona Supercomputing Center, Extrae instrumentation library, (accessed June 16, 2016) [Online]. Available: https://tools.bsc.es/extrae. 2016.
[16] V. Pillet, J. Labarta, T. Cortes, S. Girona, Paraver: A tool to visualize and analyze parallel code, in: Proceedings of WoTUG-18: Transputer and Occam Developments, volume 44, 1995, pp. 17–31.
[17] J. Balart, A. Duran, M. Gon, X. Martorell, E. Ayguadé, J. Labarta, Nanos mercurium: a research compiler for openMP, in: Proceedings of the European Workshop on OpenMP, volume 8, 2004, p. 56.
[18] A. Duran, R. Ferrer, E. Ayguadé, R.M. Badia, J. Labarta, in: A Proposal to Extend the openMP Tasking Model with Dependent Tasks, 37, 2009, pp. 292–305.
[19] J. Bueno, X. Martorell, R.M. Badia, E. Ayguadé, J. Labarta, Implementing OMPSs support for regions of data in architectures with multiple address spaces, in: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing, ACM, 2013, pp. 359–368.
[20] D. Bonachea, Gasnet specification, v1.1, 2002.
[21] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, Version 3.0., 2016 (accessed June 16, 2016). [Online]. Available: http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf. 2016.
[22] A.E. Eichenberger, J. Mellor-Crummey, M. Schulz, M. Wong, N. Copty, R. Dietrich, X. Liu, E. Loh, D. Lorenz, OMPT: An openMP tools application programming interface for performance analysis, in: OpenMP in the Era of Low Power Devices and Accelerators, Springer, 2013, pp. 171–185.
[23] B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, O. Shivers, The flux OSKit: A substrate for OS and language research., in: In Proceedings of the 16th ACM Symposium on Operating Systems Principles, Saint-Malo, France, October 1997., 1997, pp. 38–51.
[24] FLUX Research Group, OSKit List Memory Manager, (accessed May 17, 2017). [Online]. Available: https://www.cs.utah.edu/flux/oskit/html/oskit-wwwch25.html. (2012).
[25] M. Masmano, I. Ripoll, P. Balbastre, A. Crespo, A constant-time dynamic storage allocator for real-time systems., Real Time Syst. 40 (2) (2008) 149–179.
[26] C. Silvano, W. Fornaciari, G. Palermo, V. Zaccaria, F. Castro, M. Martinez, S. Bocchio, R. Zafalon, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, M. Wouters, C. Kavka, L. Onesti, A. Turco, U. Bondik, G. Mariani, H. Posadas, E. Villar, C. Wu, F. Dongrui, Z. Hao, T. Shibin, Multicube: Multi-objective design space exploration of multi-core architectures, in: VLSI 2010 Annual Symposium, Springer, 2011, pp. 47–63.
[27] R. Giorgi, R.M. Badia, F. Bodin, A. Cohen, P. Evripidou, P. Faraboschi, B. Fechner, G.R. Gao, A. Garbade, R. Gayatri, S. Girbal, D. Goodman, B. Khan, S. Koliaï, J. Landwehr, N.M. Lê, F. Li, M. Lujànj, A. Mendelson, L. Morin, N. Navarro, T. Patejko, A. Pop, P. Trancoso, T. Ungerer, I. Watson, S. Weis, S. Zuckerman, M. Valero, TERAFLUX: harnessing dataflow in next generation teradevices, ELSEVIER Microprocess. Microsyst. 38 (8, Part B) (2014) 976–990.
[28] R. Giorgi, Exploring future many-core architectures: The TERAFLUX evaluation framework, Advances in Computers, Elsevier, 2016.
[29] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, D. Ortega, COTSOn: infrastructure for full system simulation, ACM SIGOPS Oper. Syst. Rev. 43 (1) (2009) 52–61.
[30] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, N.P. Jouppi, McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures, in: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, ACM, 2009, pp. 469–480.
[31] R. Giorgi, P. Faraboschi, An introduction to DF-threads and their execution model, in: Computer Architecture and High Performance Computing Workshop (SBAC-PADW), 2014 International Symposium on, IEEE, 2014, pp. 60–65.
[32] R. Giorgi, A. Scionti, A scalable thread scheduling co-processor based on data-flow principles, Fut. Gener. Comput. Syst. 53 (2015) 100–108.
[33] R. Giorgi, Teraflux: exploiting dataflow parallelism in teradevices, in: Proceedings of the 9th conference on Computing Frontiers, ACM, 2012, pp. 303–304.
[34] UDOO, UDOOX86: The Most Powerful Maker Board Ever, (accessed May 17, 2017).[Online]. Available: https://www.kickstarter. (2016).
[35] A. Rizzo, G. Burresi, F. Montefoschi, R. Giorgi, Making iot with UDOO, Interact. Des. Arch. (2016) 95–112.
[36] R. Giorgi, Scalable embedded systems: Towards the convergence of high-performance and embedded computing, in: Embedded and Ubiquitous Computing (EUC), 2015 IEEE 13th International Conference on, IEEE, 2015, pp. 148–153.
[37] J. Shi, J. Wan, H. Yan, H. Yan, A survey of cyber-physical systems, in: Wireless Communications and Signal Processing (WCSP), 2011 International Conference on, IEEE, 2011, pp. 1–6.
[38] L. Jozwiak, M. Lindwer, R. Corvino, P. Meloni, L. Micconi, J. Madsen, E. Diken, D. Gangadharan, R. Jordans, S. Pomata, P. Pop, G. Tuveri, L. Raffo, G. Notarangelo, ASAM: Automatic architecture synthesis and application mapping, Microprocess. Microsyst. 37 (8) (2013) 1002–1019.
[39] F. Bernier, J. Ploennigs, D. Pesch, S. Lesecq, T. Basten, M. Boubekeur, D. Denteneer, F. Oltmanns, F. Bonnard, M. Lehmann, T.L. Mai, A.M. Gibney, S. Rea, F.A.o. Pacull, C. Guyon-Gardeux, L.-F. Ducreux, S. Thior, J. Verriet, M. Hendriks, S. Fedor, Architecture for self-organizing, co-operative and robust building automation systems, in: Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE, IEEE, 2013, pp. 7708–7713.

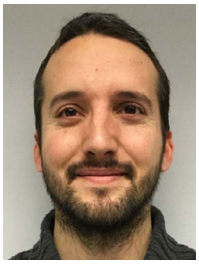
Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

Dimitris Theodoropoulos obtained his Diploma (5-year degree) and M.Sc degree respectively from the Electronic and Computer Engineering department at the Technical University of Crete, Greece. In 2007, he joined the Computer Engineering department of the Delft University of Technology, the Netherlands, where he received his PhD. In 2011, he joined the Computer Architecture and VLSI Systems group at the Foundation for Research and Technology - Hellas (FORTH) in Greece, where he is working as a post-doc researcher for national and international research projects. His research interests are in the domains of Embedded Systems, Computer Architecture, and Reconfigurable computing.

Somnath Mazumdar is a PhD student at the Department of Information Engineering, University of Siena, Italy. His main research interests are Dataflow-based computing, (heterogeneous) computer architectures (mainly processor architectures, interconnects), Cloud computing, resource allocation, and performance analysis. In the past, he consulted on a variety of projects, involving qualitative and quantitative analysis of Cloud, Grid computing.

Eduard Ayguadé received the Engineering degree in Telecommunications in 1986 and the Ph.D. degree in Computer Science in 1989, both from the Universitat Politècnica de Catalunya (UPC), Spain. Since 1987 Prof. Ayguadé has been lecturing at the Computer Science School (FIB) and Telecommunications Engineering (ETSETB) both in Barcelona. Currently, and since 1997, he is full professor of the Computer Architecture Department at UPC. Prof. Ayguadé has lectured a number of (undergraduate and graduate) courses related with computer organization and architecture, parallel programming models and their implementation. Prof. Ayguadé is also involved in the Computer Architecture and Technology PhD Program at UPC, where he has (co-)advised more than 20 PhD thesis, in topics related with his research interests: multicore architectures, parallel programming models and their architectural support and compilers for HPC architectures. In these research topics, Prof. Ayguadé has published more than 300 papers and participated in several research projects in the framework of the European Union and research collaborations with companies related with HPC technologies (IBM, Intel, Nvidia, Microsoft and Samsung). Currently Prof. Ayguadé is associated director for research on the Computer Sciences Department at the Barcelona Supercomputing Center (BSC-CNS), the National Center for Supercomputing in Spain located in Barcelona.

Nicola Bettin earned his B.S degree in Electronic Engineering at University of Padua and in 2011 he obtained his M.S degree in Electronic Engineering at University of Bologna. In 2012 he joined the Technology Transfer Team T3LAB, in Bologna, and co-founded the FPGA Group. He did research in the design of a standard HW/SW architecture for machine vision and developed commercial solutions for processing multimedia data stream in embedded systems. His main interests were FPGA solutions and heterogenic multi-core system-on-chip solutions. He joined the electronic R&D dept. at Vimar Group in 2015 and his research activity is mainly focused on human interaction with smart home systems.



Javier Bueno Hedo holds a PhD. degree in Computer Science from the Technical University of Catalonia (UPC). He became involved in research in 2004, when he started as a part-time student in the European Center of Parallelism of Barcelona (CEPBA) working with Software Distributed Memory Systems. In 2006 he became a full-time junior researcher at the Barcelona Supercomputing Center (BSC) and continued his work on distributed systems. From 2010 to 2015 he worked on his thesis, which provided the OmpSs programming model with support for clusters of multi-cores and clusters of GPUs. This work has also been applied to different research projects such as the Mont-Blanc2 project. His current research aims to produce new programming models and tools to ease the complexity of developing applications for modern HPC systems.

Sara Ermini has a Master's degree in Persuasive Communication and New Media at the University of Siena (Italy) where she is now holding a research position in Scenario Based Design for the H2020 European AXIOM Project. Her research interests include IXD, UxD and Interactive Machine Learning.

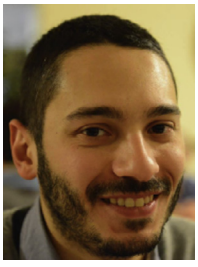
Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

Antonio Filgueras received a degree in computer science at Universitat Politècnica de Catalunya - BarcelonaTech (UPC) in 2012. Currently working at the Programming Models group of Barcelona Supercomputing Center and participating in the AXIOM European project. His research interests are focused on heterogeneous and reconfigurable solutions for high performance computing and programmability of those.

Dr. Daniel Jiménez-González received the M.S. and Ph.D. degrees in Computer Science from the Technical University of Catalunya (UPC) in 1997 and 2004, respectively. He currently holds a position as Tenured Assistant Professor in the Computer Architecture Department at UPC, BarcelonaTech, and is a associated researcher at the Computer Sciences-Programming Models Department at BSC-CNS. His research interests cover the areas of parallel architectures, runtime systems, compilers and reconfigurable solutions for high-performance multiprocessor systems. Dr. Jimenez-Gonzalez has coauthored more than 40 publications in international journals and conferences. He is currently co-advising 1 PhD student and has co-advised 2 PhD students. He has been participating in the HiPEAC Network of Excellence and in the SARC, ACOTES, TERAFLUX, AXIOM and PRACE European projects.

Carlos Álvarez received the M.S. and Ph.D. degrees in Computer Science from the Technical University of Catalunya (UPC) in 1998 and 2007, respectively. He currently holds a position as Tenured Assistant Professor in the Computer Architecture Department at UPC, BarcelonaTech, and is an associated researcher at the Computer Sciences-Programming Models Department at BSC-CNS. His research interests cover the areas of parallel architectures, runtime systems and reconfigurable solutions for high-performance multiprocessor systems. He has coauthored more than 40 publications in international journals and conferences. He is currently advising 1 PhD student and has co-advised 2 PhD theses. He has been participating in the HiPEAC Network of Excellence and in the TERAFLUX and AXIOM European projects.

Xavier Martorell received the M.S. and Ph.D. degrees in Computer Science from the Technical University of Catalunya (UPC) in 1991 and 1999, respectively. Since 1992 he has lectured on operating systems, parallel runtime systems and OS administration. He has been an associate professor in the Computer Architecture Department at UPC since 2001. His research interests cover the areas of operating systems, runtime systems, compilers and applications for high-performance multiprocessor systems. Dr. Martorell has participated in several long-term research projects with other universities and industries, primarily in the framework of the European Union ESPRIT, IST and FET programs. He spent one year working with the BG/L team in the IBM Watson Research Center. He has coauthored more than 60 publications in international journals and conferences. He has co-advised three Ph.D. theses and he is currently advising 3 PhD students. He is currently the Manager of the Parallel Programming Models team at the Barcelona Supercomputing Center. He has been participating in the HiPEAC Network of Excellence and in the SARC, ACOTES, and Intone, POP, ENCORE, MontBlanc (I and II), DEEP/DEEP-ER and the AXIOM European projects.



Francesco Montefoschi obtained his B.S. degree in Computer Science Engineering at UniversitÃă degli Studi di Siena in 2015. He worked as software developer in the passenger transportation domain at Allbus from 2009 to 2015. Currently is holding a research position at Università degli Studi di Siena on the H2020 AXIOM Project. He is also part of the UDOO Team, porting and optimizing the supported operating systems (Ubuntu and Android) for the UDOO boards. His interests include embedded systems, machine learning, hardware hacking, do-it-yourself, open source software, web and mobile application development.

David Oro received the B.S, and M.S. degrees in Computer Science from Universitat Politècnica de Catalunya (UPC) in 2006, and an M.S. degree in Computer Architecture in 2011, also from UPC. He started his professional career in 2005 working as a consultant in performance monitoring solutions. In 2009, he joined the Barcelona Digital Technology Centre where he held a research position on online banking cybercrime mitigation for CaixaBank. Currently, he works for Herta Security leading the GPU parallelization of several products. He has published several papers in international peer-reviewed conferences and holds two patents. His research interests include computer architecture, GPU computing and malware analysis.

Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

Dionisios Pnevmatikatos is a Professor and former Chair of the Electronic and Computing Engineering Department, Technical University of Crete and a Researcher at the Computer Architecture and VLSI Systems (CARV) Laboratory of the Institute of Computer Science, FORTH in Greece. He received his B.Sc. degree in Computer Science from the Department of Computer Science, University of Crete in 1989 and M.Sc. and Ph.D. degrees in Computer Science from the Department of Computer Science, University of Wisconsin-Madison in 1991 and 1995 respectively. His research interests are in the broader area of Computer Architecture, where he investigates the Design and Implementation of High-Performance and Cost-Effective Systems, Reliable System Design, and Reconfigurable Computing.

Antonio Rizzo Full Professor of Interaction Design, Universitá di Siena and Co-founder UDOO (Present). Director Academy of Digital Arts and Science' - ArsNova (2000–2009). Chair of the European Association of Cognitive Ergonomics (2000–2006). Member di WG30 NATO Human Factors and Human Reliability Group (1999 - 2002). Member of the Programme Incitatif de Recherche sur l'Education et la Formation (PIREF) of the French Government (2002–2003). Head of the Human Factor Group of the Italian National Railways (1996–1999). Liaison for Apple Inc. for the Apple Design Project (1996–1997).

Dr. Paolo Gai, CEO, graduated (cum laude) in Computer Engineering at University of Pisa in 2000 with a graduation thesis developed at the ReTiS Laboratory of the Scuola Superiore Sant'Anna on the development of the modular real-time kernel SHaRK. He obtained the PhD from Scuola Superiore Sant'Anna in 2004. Since 2000, he founded the ERIKA Enterprise project, an open-source RTOS which recently reached the OSEK/VDX certification, and which is currently used by various industries and universities. Since 2002 he is CEO and founder of Evidence Srl, a SME working on operating systems and code generation for Linux- and ERIKA- based industrial products in the automotive and white goods market. Since 2011 he is President and founder of SSG Srl, providing hardware turnkey solutions for the white goods market. His research interests include development of hard real-time architectures for embedded control systems, multi-processor systems, object-oriented programming, real-time operating systems, scheduling algorithms and multimedia applications.

Stefano Garzarella is a Software Engineer with a strong experience in kernel and user space programming, especially in the virtualization and networking fields. He has been a FreeBSD and Linux developer since 2013. He graduated (summa cum laude) in Computer Engineering at the University of Pisa in 2014 for a thesis entitled "Optimization of the FreeBSD network stack for high-speed networks". After the thesis, he collaborated with Prof. Luigi Rizzo for developing a virtual netmap passthrough for VMs (ptnetmap) and other improvements for netmap (a framework for high-speed packet I/O). He successfully participated at the Google Summer of Code 2015 to develop a FreeBSD support for ptnetmap. Currently, he works for Evidence SRL on Linux kernel, networking, and virtualization in embedded systems.

Dr. Bruno Morelli received Master Degree in Computer Engineering at University of Pisa in 2007. He is expert of the whole Linux ecosystem, from the development of Linux kernel drivers to the integration and modification of libraries and development tools. He has a good knowledge of C/C++ languages, bash scripting, ARM and x86 assembler. He works as Linux Software Engineer at Evidence Srl since 2007.

Alberto A. Pomella, Electronics & Software R&D Manager of Vimar S.p.A., Standalone and Home and Building Automation products (present). R&D Director at CRS (2001–2003), Home automation Products; UX and embedded PC development group at SELCA S.p.A. (1992–2001); Project Validation Group for consumer PC at ASEM (1991–1992). Degree in Electronic Engineering from Politecnico of Torino in 1990, with specialization in software development and industrial automation.

Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018

Roberto Giorgi is an Associate Professor at Department of Information Engineering, University of Siena, Italy. He was Research Associate at the University of Alabama in Huntsville, USA. He received his PhD in Computer Engineering and his Master in Electronics Engineering, Summa cum Laude both from University of Pisa, Italy. He is the coordinator of the European Project AXIOM. He coordinated the TERAFLUX project in the area of Future and Emerging Technologies for Teradevice Computing. He is participating in the European projects HiPEAC (High Performance Embedded-system Architecture and Compiler), ERA (Embedded Reconfigurable Architectures). He contributed to SARC (Scalable ARChitectures), ChARM (performance evaluation of ARM-processor based embedded systems). His current interests include Computer Architecture themes such as Embedded Systems, Multiprocessors, Memory System Performance, Workload Characterization.

Please cite this article as: D. Theodoropoulos et al., The AXIOM platform for next-generation cyber physical systems, Microprocessors and Microsystems (2017), http://dx.doi.org/10.1016/j.micpro.2017.05.018