Lagarto I RISC-V Multi-Core: Research Challenges to build and integrate a Network-on-Chip

Neiel I. Leyva-Santes[1], Ivan Pérez[2], César A. Hernández-Calderón[1], Enrique Vallejo[2], Miquel Moretó[3], Ramón Beivide[2], Marco A. Ramírez-Salinas[1], Luis A. Villa-Vargas[1].

Israel.leyva.santes@gmail.com, ivan.perezgallardo@unican.es, hdzces@gmail.com, enrique.vallejo@unican.es, miquel.moreto@bsc.es, ramon.beivide@unican.es, {mars,lvilla}@cic.ipn.mx.

[1]Computing Research Center, National Polytechnic Institute. Mexico City, Mexico. [2]University of Cantabria. Santander, Spain. [3]Barcelona Supercomputing Center. Barcelona, Spain.

Abstract. Current compute-intensive applications largely exceed the resources of single-core processors. To face this problem, multi-core processors along with parallel computing techniques have become a solution to increase the computational performance. Likewise, multi-processors are fundamental to support new technologies and new science applications challenges. A specific objective of the Lagarto project developed at the National Polytechnic Institute of Mexico is to generate an ecosystem of high-performance processors for the industry and HPC in Mexico, supporting new technologies and scientific applications. This work presents the first approach of the Lagarto project to the design of multi-core processors and the research challenges to build an infrastructure that allows the flagship core of the Lagarto project to scale to multi- and many-cores. Using the OpenPiton platform with the Ariane RISC-V core, a functional tile has been built, integrating a Lagarto I core with memory coherence that executes atomic instructions, and a NoC that allows scaling the project to many-core versions. This work represents the initial state of the design of mexican multi-and many-cores processors.

Keywords: Multi- and Many-core, Multiprocessors RISC-V, Interconnection networks.

1 Introduction

The "Lagarto" project [1], developed by the Computer Architecture research team at the Computing Research Center of the National Polytechnic Institute of Mexico, aims to generate an open computing platform for academia and research to ease the understanding of fundamental concepts of Computer Architecture and Operating Systems. The project has two main branches of development: the first one is focused on the educational area, and the second one on the development of a high-performance processors ecosystem, targeting both industry and HPC.

Around 2006, multi-core processors appeared in the market to overcome the "power wall". Multi-core processors increase compute power via parallelism, which rises dramatically the complexity of microprocessors design and programming.

Considering the HPC segment, nowadays all systems rely on multi-core processors, often coupled with highly-parallel accelerators. The use of multiple interconnected compute nodes, each of them with several multi-core processors, allows to build supercomputers with high performance. HPC systems have been built with more than 10 million cores (for example, the Sunway TaihuLight of the National Supercomputing Center in Wuxi) and peak performance above 100,000 TFLOPs (for example The Summit of the Oak Ridge National Laboratory). Relevant information can be found in the Top500 list [3].

The first core designed in the Lagarto project is denoted Lagarto I. The Lagarto I core design supports different ISAs, including RISC-V [2]. RISC-V is an instruction set architecture (ISA) that is completely open and freely available to industry, license-free and royalty-free. RISC-V is a RISC-based architecture; RISC-based processors lead the mobile and embedded markets, partially because of their characteristics of low energy consumption and small area. By contrast, the HPC market is nowadays dominated by traditional x86 processors. However, a growing share of HPC systems rely on the use of RISC designs. Some examples are the BlueGene systems, the Mont-Blanc project and the Tibidabo cluster prototype, to name a few [4][5][6] [7].

In the Lagarto project, the main challenge to overcome in the second branch (industry and HPC) is to design and develop an infrastructure that allows our processors to scale to multi- and many-cores. Such infrastructure is required to optimize and develop multi-core processors for the industry segment and many-core processors for HPC. The Lagarto project currently has two different core designs: the Lagarto I, which is a scalar core, and the Lagarto II, which is a superscalar core, both with a memory system that allows them to function as mono-core processors. To implement the parallelism required for HPC systems, it is necessary that the cores of Lagarto project can operate as multi-core system, this is, multiples core running multiple coordinated tasks in the same memory space. Scaling to many-cores requires to have a Network-on-Chip (NoC) that allows to transport messages and data between the cores and the memory system. The first objective of this stage of the Lagarto project is to develop the infrastructure required to implement many-cores processors, integrating a NoC into the memory system of Lagarto I core to allow scale to many-core.

This work shows the initial process to integrate the Lagarto I core (the educational version) into a multi-core system based on a Network-on-Chip. The design relies on a RISC-V development framework. We analyze the current RISC-V open-source ecosystem and discuss the selection of one of the available development frameworks, in order to select the most compatible with the Lagarto I core. The analysis of previous works between RISC-V and these platforms with NoCs were fundamentals to select a platform; Also, we show the final design of "Lagarto I" tile that we will use in our versions of multi and many cores.

The rest of the work is organized as follows. Section 2 presents the required background, including Atomic instructions, Cache coherence protocols, interconnection networks and relevant development tools. Section 3 presents an overview of the Lagarto Project and the architecture of the educational core "Lagarto I". Section 4 discusses the selection of the tools to integrate and design the Lagarto I tile. Section 5 presents the fundamental modules of the architecture of Lagarto I tile. Section 6 discusses the advantages and disadvantages of Lagarto I tile final design, possible optimizations and the future work.

2 Background

This section presents the required background to understand the paper. First, the main fundamentals related to the design of multi-core processors are presented. Next, it discusses relevant development tools, covering both simulation tools and hardware development frameworks.

Shared-memory multi-core CPU design mainly involves three aspects:

a) Atomic instructions: used to atomically access data that is shared between cores. They are required to build synchronization mechanisms like mutex, semaphores or locks.

b) Cache coherence protocols and memory consistency: essential to maintain coherence and consistency among shared data hosted in the private caches of each core.

c) The interconnection network: to transport control messages and data between cores and the memory system.

Despite dealing with different aspects of the design, they must be considered as a whole when designing the complete system. These aspects are discussed next.

a) Atomic instructions

During multithreaded execution, it is essential to synchronize the running subprocesses on the different threads. This is usually solved by executing "read-modify-write" (RMW) atomic operations. An atomic operation refers to the execution of RMW in a single apparent step. This is implemented by carrying out the read and write operations consecutively, blocking the interruptions to prevent the interleaving of any other memory access with these read and write operations. This guarantees the atomicity of the memory access [8], without interleaving other accesses.

The "A" extension of the RISC-V ISA is the subset of the architecture that describes atomic instructions. RMW atomic instructions are defined in this extension to support the synchronization of multiple RISC-V hardware threads running in the same memory space. It contains two sets of atomic instructions: load-reserved/store-conditional and atomic fetch-and-op memory instructions.

RISC-V supports several consistency models such as: unordered, acquire, release and sequential consistency (SC). SC is a highly intuitive consistency model introduced by L. Lamport [9], who formulated that a multiprocessor is sequential consistent if it complies with the following: a) the result of any execution distributed in multiple cores must be the same as when all the executed instructions are run in a certain sequential order, and b) operations assigned to a specific kernel must maintain the sequence specified by the program. However, this simplicity significantly restricts the implementation of memory units and hinders performance. More relaxed memory models allow to reorder memory accesses and to implement less restrictions in the load/store units of the cores. This often translates into higher performance, at the cost of memory access rules which are harder to reason about and the requirement of explicit fences to delimit synchronization accesses.

b) Cache coherence protocols

The cache coherence protocol distributes the updates performed on each memory location to all the private caches in the system, guaranteeing that all the cores observe, at a given point in time, the updates to shared data.

Snoopy based cache coherence protocols typically rely on broadcast mediums like buses or crossbars, where every every connected element listens to all the traffic. This kind of design offers a global vision of the memory accesses to every element interconnected, easing the implementation of the coherence protocol. However it becomes a bottleneck when it interconnects several cores, as all of them share the same network resources. This is the most common solution when the processor has a low number of cores, typically less than 10.

By contrast, directory based cache coherence protocols are typically used with distributed cache memories and NoCs. This type of design follows a distributed philosophy, in which each of several directory controllers tracks the state of the data assigned to it. In other words, each directory acts as a server to data requests that belongs to its domain. This scheme, supports parallel requests to different directories, with the NoC transporting the messages with the result of these requests. Building a distributed cache memory system implies more complexity due to the lack of global information, requiring more complex coherence protocol implementations. However, it is a more scalable solution.

c) Interconnection networks

The system networks are used in supercomputer and datacenters to interconnect multiple servers, each server with multiple cores. For each server (node) to communicate with other nodes, it can use electrical or optical cables to transfer the data packets.

Similarly, the processor internal cores, memory controllers and peripheral controllers communicate through an on-chip interconnect. Typically, when cores work in parallel, they share data in the different shared-memory levels. The interconnection serves as a communication channel to transfer data between the levels of the memory hierarchy and to maintain coherence between different copies of the same data.

Buses and crossbars are frequently employed in the interconnections in multi-core CPUs. They are simple to design and perform well for a low amount of cores. Nevertheless, they do not scale well, and alternative solutions have been used when interconnecting tens of cores, which is typical in the HPC market. These alternatives are denoted Networks on Chip (NoC).

NoC are scalable interconnects that typically rely on tile-based CPUs. Commonly, each tile has one or more cores with their private caches, plus a bank of a shared and distributed cache. Some of them have memory and peripheral controllers too. Such basic block is replicated multiple times to build a processor of the required size, including one router in each tile. Typical topologies for NoCs include rings and meshes.

Rings have been used widely, for example by Intel in their Xeon processors. More recently, Intel has included meshes in their 72-core Xeon Phi Knights Landing accelerator [10] and the newest Xeon processors. ARM also offers a mesh interconnect that supports up to 128 cores [11]. Meshes are gaining ground in the market as they are more scalable than rings, thought large designs might introduce more scalable topologies in the future, like torus or Flattened Butterflies.

d) Development tools

There are platforms dedicated to the design and simulation of multicore systems and interconnection networks. Standalone NoC simulators, such as Booksim or Garnet, typically employ traffic generators to create random requests to feed the network. However, some platforms allow to perform whole-system functional simulation at the software level, such as Spike and Gem5.

Alternatively, there are platforms that perform RTL simulations with specific cores and synthesize multicore processor designs in FPGAs, supporting parameter configuration such as number of threads and cores, cache size, interconnection type, processor, etc.

Table 1. shows some of the most popular platforms available. Some platforms offer processor designs for free, but the tools needed to work with the design are proprietary and require a license. This is the case of the Bluespec company, which offers a platform with two freely available RISC-V processors called Piccolo and Flute. These processors are designed with the high-level language Bluespec (same as the company name), and it is necessary to acquire a Bluespec license to compile the processor codes. In addition, it offers commercial platforms such as RISC-V Verification Factory and RISC-V Acceleration Factory dedicated to the integration, verification and debugging of integrated systems and FPGAs.

Other platforms are open source designs, but their tools can be open source or not and they can be part of their own platform or not. Both Pulp and OpenPiton are open source designs. They rely on a set of open source tools such as Icarus Verilog and other proprietary tools as Mentor Graphics QuestaSim. Specifically, the users can decide between the compatible tools to design, simulate and implement their system.

While some platforms are completely dedicated to RISC-V processors, some others also support other processor ISAs, such as Gem5 and OpenPiton platforms. In the case of Gem5, it employs a generic processor model which is configured with the parameters of the target processor. By contrast, OpenPiton gives some options about the processors to integrate into its platform, for example OpenSparc T1, PicoRV32 and Ariane processors.

Table 1. Popular platforms to simulation and implementation of multicore processors.

Platform	Simulation		FPGA Implementation	Processor		Open Source
	Functional	RTL		RISC-V	Other	
Spike simulator [12]	✔	-	-	✔	-	✔
Gem5 simulator [13]	✔	-	-	✔	✔	✔
Bluespec [14]	-	✔	✔	✔	-	-
LowRisc [15]	-	✔	✔	✔	-	✔
Pulp [16]	-	✔	✔	✔	-	✔
OpenPiton [17]	-	✔	✔	✔	✔	✔

OpenSPARC T1 was the default core of the OpenPiton processor from the OpenPiton platform. Given the potential of the NoC and the protocols of coherency implemented in the cache hierarchy of the OpenPiton processor, in recent years, projects have emerged with the main goal of incorporating RISC-V cores into the OpenPiton processor. Some examples are Juxtapiton [18], where the PicoRV32 core was integrated with the OpenPiton platform and inherited all the capabilities of the infrastructure, and the integration of Ariane core into the OpenPiton processor [19], started in 2019 by the ETH Zürich and the Wentzlaff Parallel Research Group.

3 Lagarto I RISC-V architecture

The computer architecture design team of the Microelectronic and Embedded System Laboratory of the Computing Research Center of the National Polytechnic Institute of Mexico has developed the Lagarto I core. Its main goal is to provide its users with a scalable low-cost development platform for hardware and software systems. These users could come from the academic field as well as the industry sector. For the academic field, it will be possible for the students to understand the tradeoffs and challenges related to the microarchitecture design process; for the industry sector, it will grant the opportunity to use a customizable development platform.

Lagarto I core is a scalar 64 bit RISC-V based processor, that supports the RV64I instruction set, the M extension for multiplication and division, and the A extension for atomic operations, the minimum requirements to support an embedded Linux booting process. Additionally, Lagarto I supports the RISC-V Privileged Architecture Version 1.7.

Lagarto I core has been developed in Verilog, taking advantage of the hardware description language tools for designing, testing and verifying digital systems as robust as a SoC. The microarchitecture implements a six-stage pipeline and includes optimized modules for instruction processing. Additionally, Lagarto I includes a Memory Management Unit looking to allow not only the flow of data through the different memory levels, but also the implementation of multicore systems, establishing the guideline to explore massive processing systems for High Performance Computing.

4 Design proposal for Lagarto Multi-core

Currently, tiled architectures have gained ground in the design of multicores. They provide communication using shared memory and direct communication networks, allowing the flow of data and messages from one tile to another without the need to use system software [20].

The minimum requirements of a tile are the processor, L1 cache memory (including coherence support), private or shared L2 memory and an interconnection network. In order to have an infrastructure that allows the Lagarto I processor to scale to a large number of cores, its current design needs to be transformed to a tile implementation, which we denote the Lagarto tile. Starting from the single-core Lagarto I design, it requires a coherence protocol in the L1 cache memory, an interconnection network and a shared memory level. Likewise, it requires a chipset that hosts the memory controllers and the OS boot. It is also essential to incorporate the atomic instructions, a feature already implemented in the Lagarto I processor.

The available open-source multi-core processor design platforms have been analyzed to determine if they can be used to facilitate the design of multi-core Lagarto processors. Three open-source platforms have been chosen for the analysis: Lowrisc, PULP and OpenPiton. Table 2. shows a summary of the analysis of the three platforms under the following headings: a) HDL language, which should be compatible with that used in Lagarto I core, b) interconnection network used, c) coherence protocol support, d) compatible with RISC-V and e) OS boot support.

The chosen platforms are suitable to be integrated into this work thanks to the fact that they have been developed with HDL, as well as Lagarto I core. LowRisc, in particular, has been developed in the high-level language Chisel. With respect to the interconnection network, only OpenPiton implements a NoC. The Pulp platform does not support a coherence protocol, since its designs are focused on hardware acceleration. The three platforms integrate at least one RISC-V processor and implement atomic instructions, although Pulp only implements atomic fetch-and-op memory instructions.

OpenPiton has been selected as the most suitable platform for this work, mainly for its ability to scale the number of cores thanks to the implemented NoC. It is completely free, fully configurable (cache size, interconnection network, processor) and modular, exploiting the use of the Verilog language. It is a platform developed mainly in Verilog for hardware description and Perl + Python to integrate tools and build designs. In the most recent version, released in 2019, it supports three different processors: OpenSPARC T1 and two RISC-V based processors, PicoRV32 and Ariane, integrated from the PULP framework.

Table 2. Analysis of Development platforms to be used with Lagarto I

Platform	Developers	Language	Interconnect		Coherence Protocol	Memory Levels	Processor		Linux boot	
			Crossbar	Noc			RISC-V	Others	Mono core	Multi core
LowRisc	University of Berkeley and Cambridge	Chisel	✔	-	✔	2	✔	-	✔	-
Pulp	ETH Zürich	System Verilog	✔	-	-	2	✔	-	✔	-
OpenPiton	Princeton University	Verilog	✔	✔	✔	3	✔	✔	✔	-

5 Lagarto I Multi-core

This section presents the design decisions for the Lagarto Multi-core system. It first details the recent integration of OpenPiton with the Ariane processor. Next, it presents the implementation details required for Lagarto I.

The most recent OpenPiton release integrates the OpenPiton project with the Ariane processor. This design has been used as a reference to build and design the Lagarto I tile. This design upgrades the RISC-V privileged extension from 1.7 to 1.10; this is required to support more stable OS execution and boot, offers more support to physical-memory protection (PMP) and Virtual-memory page faults. This upgrade has been analyzed to obtain enough experience to identify, recognize and adapt the required modifications for the upgrade. This design will be considered a reference to compare the execution of instructions and the flow of control messages within the NoCs, simplifying the debug process of the design.

The original version of OpenPiton leveraged an OpenSPARC core to build a tiled NoC design. The OpenSPARC T1 processor implements 8 cores connected through a crossbar, using a write-through policy to guarantee cache coherency. However, such design does not scale to large core counts, since it relies on the centralized crossbar to serialize memory accesses and invalidate sharers.

In order to scale the system, the OpenPiton approach implements an additional cache level between the private L1 and the shared L2. This new private memory level is denoted L1.5 cache, because it sits between the original L1 and the L2. Therefore, the design employs two private cache levels per core. The first level is called L1 and is part of the OpenSPARC T1 processor, operating with its write-through policy. The second level is the L1.5, which is a write-back private data cache that implements a MESI coherence protocol, allowing larger scalability. With this approach, the OpenSPARC T1 RTL is not modified and the L1.5 operates as a higher cache level above the L1. In scalable multicores, write-back caches operate more efficiently than write-through since the high write-bandwidth required by the write-through caches produces data congestion into the NoC and the memory hierarchy [21]. The OpenPiton L1.5 cache only has one input port, where both instruction and data requests are received. Both requests can be generated simultaneously in the same processor cycle, so the design requires an arbiter to determine the priority to send each request to the L1.5. In the design, instruction fetch requests always receive higher priority.

The OpenPiton project has integrated two different processor cores with different approaches with respect to the coherence protocol. In [22] the PicoRV32 processor replaces the OpenSparc T1 in OpenPiton. In this case, the processor is directly connected to the L1.5 cache from OpenPiton, removing its original L1 cache. This represents a relevant improvement at the hardware level, because it avoids the duplication of caches, with the corresponding increase in area and latency. However, it only operates with virtual memory accesses since the L1.5 cache in OpenPiton doesn't include an MMU. In [24] the Ariane processor was integrated with OpenPiton. The implementation turns its L1 write-back cache into a write-through cache similar to the one in OpenSparc T1 L1, and adapts the interfaces between Ariane L1 write-through cache and OpenPiton L1.5 cache using an arbiter. With this change, the Ariane cache system become similar to the one used by OpenSparc T1.

Our work adapts the Lagarto I core to the OpenPiton project following a similar approach. The basic building blocks employed in our project are shown in Figure 1. Fig. 1 a) shows the design of the Lagarto tile. Each tile has a Lagarto I core, the L1 cache from the Ariane processor and the related modules. Fig. 1 b) shows the OpenPiton structure, including caches, NoC routers and chipset.

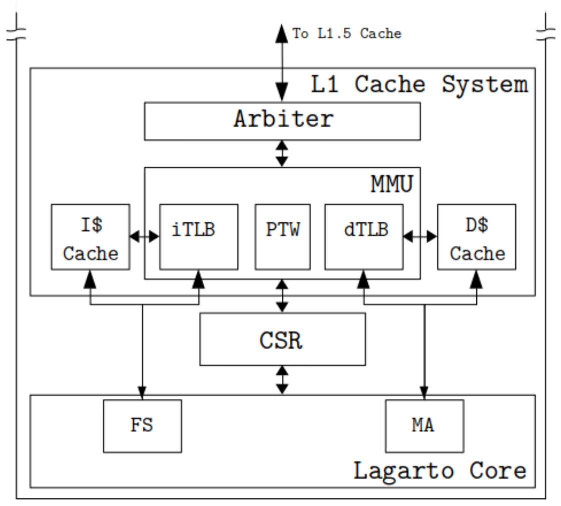
Lagarto I core interacts with the L1 cache in its Fetch Stage (FS) to request instructions, and in its Memory Access stage (MA) to load/stores data in memory. It also interfaces with the Control and Status Registers (CSR) which are read or written by privileged instructions. The upgrade of the privilege extension to version 1.10 requires to add and discard some registers within the CSR module, as well as adding hardware support for new instructions into the Lagarto core microarchitecture.

Originally, the Lagarto I processor implements a Memory Management Unit (MMU) that includes L1 caches for data and instructions, with the L1 data cache operating with a write-back policy. Considering the alternative approaches in previous works, the approach followed to merge with OpenPiton was similar to the most recent case of Ariane. The Lagarto MMU is preserved, to access the L1 caches using physical addresses. The L1 data cache is converted to write-through and connected to the L1.5 from OpenPiton. When adapting the interfaces, the alternatives are to adapt the L1 interface from Lagarto to the L1.5, or employ the caches from Ariane. We decided to employ the L1 caches from Ariane because they are already adapted to the bus arbiter interface and the L1.5. This design is suboptimal because it almost replicates the caches in levels L1 and L1.5, with the corresponding increase in area and latency. However, it allows for a faster design and a simpler integration with the other modules from OpenPiton, leaving the integration of both cache levels for future work.

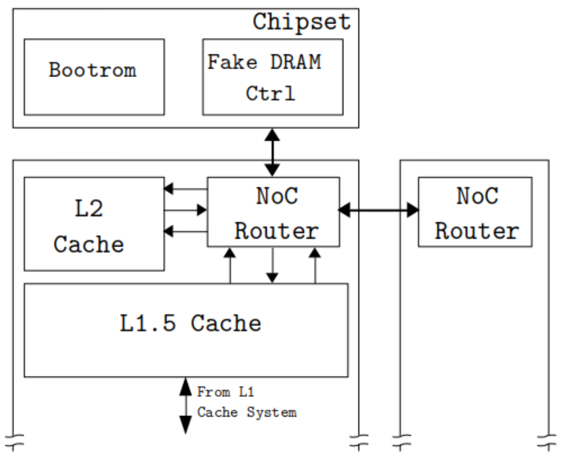
The final design of L1 cache system of the Lagarto tile integrates an instruction and data cache, MMU and an arbiter. The instruction cache has 16KB size and 4-way associativity, and the data cache has 8KB, 4-way associativity and a write-through policy for compatibility with OpenPiton. The main purpose of the MMU is to implement address translation and monitor memory accesses, to detect memory access faults. The MMU hosts the instruction and data Translation Lookaside Buffers (TLBs) and the Page Table Walker (PTW). The PTW is a hardware used to handle page faults: it listens to all incoming translation requests on both iTLB and dTLB, and in case of fault page on the TLB it will use the virtual address to start the search for its page translation in a table walk.

The module Arbiter has two main goals; first, it arbitrates between requests from the FS or MA processor stages, with FS requests always receiving more priority; second, it generates and manages the request signals, as well as packages the request to be sent to the L1.5, using an interface that is compatible with OpenPiton L1.5 cache. Likewise, the arbiter attends all responses from the L1.5 cache to the L1 cache system.

The L1.5, L2 cache and NoC router are reused from the OpenPiton project [21]. The remainder of the chipset has been modified, only reusing the bootrom module and a fake memory DRAM controller. The bootrom module contains a minimal boot of the linux code and the fake DRAM controller simulates the operation of a DRAM controller. This fake controller simulates that the operating system loads the program code into the DRAM to be executed by the cores, but the program is actually loaded manually in this fake DRAM. In the current status of the design, the program code is placed into this module to be executed after loading the boot.



a)



b)

Fig. 1. Lagarto tile a) Lagarto core connect to system cache of Ariane processor. b) Cache system, NoC, routers and chipset from Open Piton.

Figure 2. shows the block diagram of a multi-core architecture where the Lagarto tile is interconnected to others tiles through NoC routers in the P-Mesh by OpenPiton. OpenPiton provides scalable meshes of tiles on a single chip using, what they have called, a P-Mesh cache coherence system, which is capable to connect together up to 8192 chips [23]. The NoC routers are also used to interconnect the Lagarto tiles to a chipset to access the bootrom and the fake DRAM controller.

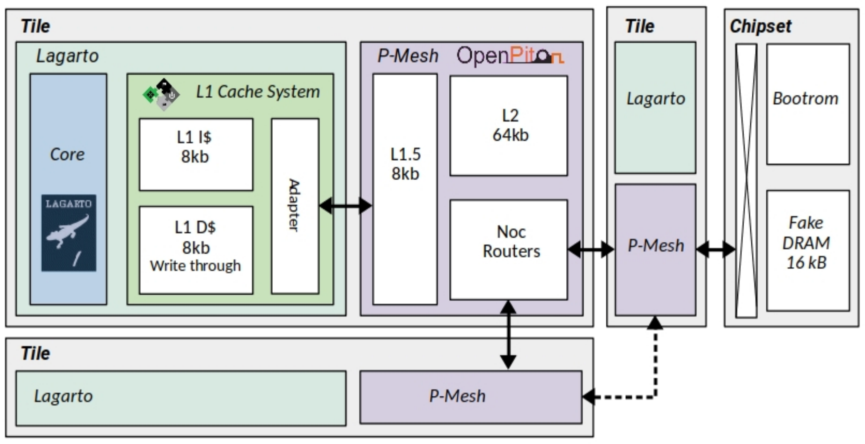


Fig. 2. block diagram of a multi-core architecture with Lagarto tiles (based on [22])

6 Conclusions and future work

This work describes the initial process of adapting the Lagarto I core to a multicore system using a NoC. The Lagarto project wants to generate a high-performance ecosystem based on RISC processors for the industry and the academic field in Mexico. For this goal, is necessary to develop an infrastructure that allows the Lagarto I processor to scale to multi- and many-cores. It has been decided to take advantage of several previous developments on multicore, which has led to perform various analysis of compatibility with the Lagarto I processor, as well as requirements for its adaptation. Consequently, the OpenPiton platform has been chosen to employ the NoC it integrates as well as the MESI protocol implemented in its memory system.

For this first approach, it has been decided to follow the process of development of previous projects carried out with processors based on RISC-V and the OpenPiton platform, which involves modifying the L1 cache to operate as a write-through cache and connect it directly with the OpenPiton L1.5 cache and thus be able to exploit the MESI protocol and access the NoC.

While effective and simpler to develop, this results in a suboptimal design because of the redundant modules. To achieve the main objective of the Lagarto project, it is desired to eventually get rid of the L1.5 memory and implement the MESI protocol in the L1 write-back memory integrated in the MMU of the Lagarto I processor. This will establish direct communication between the interface of the OpenPiton NoC to the Lagarto L1 cache, allowing direct access from the L1 cache to the L2 cache and the Chipset, as well as reducing the latency and area of the tile. An additional advantage is to be able to process instruction requests and data accesses in parallel. Finally, it will be necessary to perform tests to guarantee memory consistency and to successfully execute a SMP Linux boot.

References

[1] Ramírez, C., Hernández, C., Morales, C.R., García, G.M., Villa, L.A., & Ramírez, M.A. (2017). Lagarto I - Una plataforma hardware/software de arquitectura de computadoras para la academia e investigación. Research in Computing Science, 137, 19-28.
[2] “The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2”, Editors Andrew Waterman and Krste Asanović, RISC-V Foundation, May 2017.
[3] TOP500 List - November 2018 | TOP500 Supercomputer Sites. (2019). Retrieved from https://www.top500.org/list/2018/11/
[4] N. R. Adiga et al., "An Overview of the BlueGene/L Supercomputer," SC '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, Baltimore, MD, USA, 2002, pp. 60-60. doi: 10.1109/SC.2002.10017
[5] IBM100 - Blue Gene. (2012). Retrieved from https://www.ibm.com/ibm/history/ibm100/us/en/icons/bluegene/
[6] N. Rajovic et al., "The Mont-Blanc Prototype: An Alternative Approach for HPC Systems," SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, 2016, pp. 444-455. doi: 10.1109/SC.2016.37
[7] Rajovic, N., Rico, A., Puzovic, N., Adeniyi-Jones, C., & Ramirez, A. (2014). Tibidabo: Making the case for an ARM-based HPC system. Future Generation Computer Systems, 36, 322-334. doi: 10.1016/j.future.2013.07.013
[8] Sorin, D. J., Hill, M. D., Wood, D. A., 2011. A Primer on Memory Consistency and Cache Coherence, 1st Edition. Morgan & Claypool Publishers.
[9] L. Lamport. How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs. IEEE Transactions on Computers, C-28(9):690–91, Sept. 1979. doi:10.1109/TC.1979.1675439
[10] A. Sodani et al., "Knights Landing: Second-Generation Intel Xeon Phi Product," in IEEE Micro, vol. 36, no. 2, pp. 34-46, Mar.-Apr. 2016. doi: 10.1109/MM.2016.25
[11]Ltd., A. System IP | CoreLink CMN-600 – Arm Developer. Retrieved from https://developer.arm.com/ip-products/system-ip/corelink-interconnect/corelink-coherent-mesh-network-family/corelink-cmn-600
[12] riscv/riscv-isa-sim. (2017). Retrieved from https://github.com/riscv/riscv-isa-sim
[13] gem5. Retrieved from http://gem5.org/Main_Page
[14] RISC-V Processor IP & Tools for Cores & Subsystems | Bluespec. (2019). Retrieved from https://bluespec.com/
[15] lowRISC · lowRISC. Retrieved from https://www.lowrisc.org/
[16] PULP platform. Open-source efficient RISC-V architecture. Retrieved from https://www.pulp-platform.org/
[17] Princeton Parallel Group. Openpiton open source research processor, 2017. http://parallel.princeton.edu/openpiton/index.html.
[18]Katie Lim, Jonathan Balkind, and David Wentzlaff. Juxtapiton: Enabling heterogeneous-isa research with RISC-V and SPARC FPGA soft-cores. CoRR, abs/1811.08091, 2018.
[19]PrincetonUniversity. Openpiton + ariane - preliminary support for ariane rv64imac core. https://github.com/PrincetonUniversity/openpiton/tree/openpiton-dev#preliminary-support-for-ariane-rv64imac-core.

[20] D. Wentzlaff et al., "On-Chip Interconnection Architecture of the Tile Processor," in IEEE Micro, vol. 27, no. 5, pp. 15-31, Sept.-Oct. 2007. doi: 10.1109/MM.2007.4378780
[21] Balkind, J., Liang, X., Matl, M., Wentzlaff, D., McKeown, M., & Fu, Y. et al. (2016). OpenPiton: An Open Source Manycore Research Framework. Proceedings Of The Twenty-First International Conference On Architectural Support For Programming Languages And Operating Systems - ASPLOS '16. doi: 10.1145/2872362.2872414
[22] Lim, Katie & Balkind, Jonathan & Wentzlaff, David. (2018). JuxtaPiton: Enabling Heterogeneous-ISA Research with RISC-V and SPARC FPGA Soft-cores.
[23] Balkind, J., Lavrov, A., McKeown, M., Fu, Y., Nguyen, T., Shahrad, M., ... & Jackson, P. J. OpenPiton: An Emerging Standard for Open-Source EDA Tool Development.
[24]Schaffner, M., & Balkind, J. (2019). OpenPiton+Ariane Tutorial. Presentation, HiPEAC 2019, Valencia. Retrieved from https://www.pulp-platform.org/docs/hipeac/openpiton_ariane_hipeac_tutorial.pdf