Definition and Evaluation of Anisotropic SoC Architectures for the Throughput Optimisation of Streaming Applications

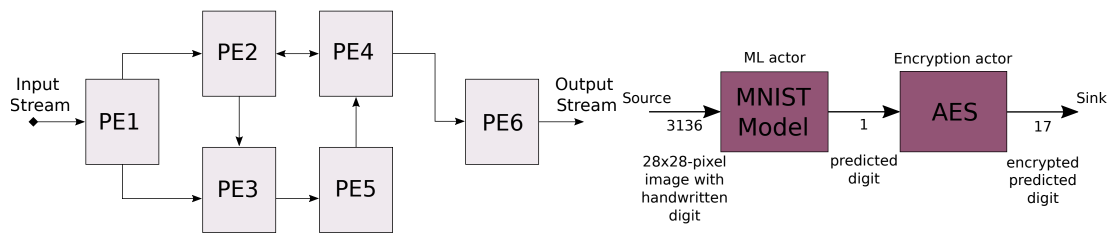
Joseph W. FAYE[1], Lucas ESTEVES ROCHA[1], Florent KERMARREC[3], Kévin MARTIN[4], Shuvra BHATTACHARYYA[12], Jean-Francois NEZAN[1], Maxime PELCAT[1]

[1] Univ Rennes, INSA Rennes, CNRS, IETR – UMR 6164, F-35000 Rennes, France.
[2] University of Maryland, College Park, USA.
[3] Enjoy-Digital, Landivisiau, France.
[4] Univ. Bretagne-Sud, Lab-STICC - UMR CNRS 6285, Lorient, France.

Abstract
Designing devices for the computing continuum, entangling near-sensor, edge, and High Performance Computing (HPC) systems, involves placing computation close to sensors to reduce data movements. Smart cameras, for example, integrate image sensors and High Performance Embedded Computing (HPeC) systems for real-time application-specific information extraction from image streams. However, energy and throughput constraints pose challenges as workloads become more complex, requiring heterogeneous Systems-on-a-Chip (SoCs) with different types of Processing Elements (PEs). Dataflow Models of Computation (MoCs) are helpful for modeling and optimizing stream processing systems. Still, current SoC architectures are designed assuming Uniform Memory Access (UMA), making it challenging to optimize coarse-grain data movements. This paper introduces the concept of anisotropic SoCs. Anisotropy is the property of an object to assume different properties in different directions, generalizing pipeline architectures and systolic arrays. Anisotropic SoCs favor specific data transfer directions in computing architectures abstracted at the system level. The paper also presents observations from an open hardware smart camera demonstrator, showing the non-trivial relationship between system anisotropy and throughput, even in a small system.

1. Introduction

Embedded systems in the Internet of Things (IoT) provide real-time access to sensors, actuators, and processing power. Smart cameras, which integrate image sensors with computer vision algorithms for processing video streams, aim to reduce data storage and transfers by computing near the sensors [15]. With the increasing integration of machine learning algorithms since 2012, optimizing machine learning algorithms for intelligent cameras has become a timely topic. Stream processing applications, such as those in smart cameras, can benefit from specialized SoC architectures, posing challenges in design space exploration for system performance [13]. While methods exist to model applications, explore the design space, and generate code, fewer efforts have been focused on accurate and early modeling of hardware architectures without complete system prototyping or simulation [11]. This paper aims to define the concept of SoC anisotropy and study its impact on multi-core hardware architectures using a canonical smart camera example with machine learning and cryptography. In the context of SoC design, anisotropy implies that some data transfer directions are favored, leading to more efficient implementation or operation, compared to others when applied to computing architectures. Source (SRC) and Sink (SNK) of data are abstracted at the system level, isolated, and implemented by scratchpad memories. This paper demonstrates that anisotropy can refer to either a computational hardware substrate or a Model of Architecture (MoA). After discussing related works in Section 2, heterogeneity and anisotropy are discussed in Section 3. Then, the experimental setup is detailed in Section 4 to demonstrate that anisotropy is an essential property for throughput optimization. Section 5 is devoted to experimental results. Finally, Section 6 concludes the paper.



(a) A coarse grain anisotropic datapath. (b) A canonical smart camera application.

FIGURE 1 – Examples of Architecture and Synchronous DataFlow (SDF) models

2. Related work

Streaming applications, such as Signal Processing and Machine Learning (SPML), require exploiting data, task, and pipeline parallelism to optimize memory and hardware resource usage. Coarse-grained data management has been central to pioneering pipelined dataflow architectures in the 1990s [2, 8] for building PEs and interconnect architectures optimized for such applications.

On one hand and due to the interest of Graphics Processing Units (GPUs), on-chip parallel programming has been dominated by bulk synchronous approaches [4]. This approach is adapted when computation occurs in super steps alternating between global parallel computing, data exchanges, and synchronization steps, leading to GPU-specific MoCs and compiling infrastructures. On the other hand, dataflow MoCs decompose processing into self-contained actors that communicate through First-In-First-Out (FIFO) queues, triggering processing upon data arrival. Data in dataflow MoCs flows through directed edges, forming a general data flow from data SRCs to data SNKs [7]. Dataflow MoCs are architecture-independent, efficient to model streaming applications, and they enable automated optimization [6].

In 2017, the stream dataflow acceleration machine was introduced [9] proving the relevance of coarse-grained reconfigurable architectures that exploit the parallelism of steaming applications. Recent open hardware initiatives provide opportunities for the design of such an architecture [5]. This paper introduces the notion of anisotropy for the design and programming of coarse-grained reconfigurable architectures for streaming applications.

Smart cameras are advancing as near-sensor stream processing HPeC devices, facing low energy and high data rate constraints. Hardware-wise, complex heterogeneous SoCs are integrated with multi-core processors, Field-Programmable Gate Arrays (FPGAs), GPUs, and dedicated Intellectual Propertys (IPs) [3]. However, heterogeneity has been limited to mass-market devices due to complex design costs and return on investment considerations. For niche markets, designers need easier-to-program hardware architectures. Reconfigurable heterogeneous architectures are becoming more programmable, making them suitable for a wider range of smart cameras. On the algorithm side, smart cameras are migrating to a hybrid structure including both Machine Learning (ML) and *expert* algorithms based on signal and image processing. While Signal Processing (SP) tasks are often based on open source code libraries like OpenCV or FFTW, and can benefit from parallel code generation via tools like [12] and [14], ML tasks are based on algorithms provided by open-source tools such as Scikit-learn [10]. These machine-learning tools integrate advanced software and hardware code-generation capabilities. The smart camera context is illustrated in figure 1 in terms of architecture and algorithm.

3. Heterogeneity and Anisotropy of SoCs Architectures

This section defines the anisotropy property of a MoA and demonstrates how this property can assist in designing throughput-efficient architectures for streaming applications.

3.1. SoC Heterogeneity

The primary component of a SoC is the PE. In a broader sense, a possible interpretation of PE involves extracting information from a bus, processing it, and then transmitting the processed data back to the bus. This general PE concept can apply to various architectures and buses, including Direct Memory Access (DMA) units, a simplified PE form executing only load and store instructions, and bus arbiters enabling multiple masters.

For our study, we narrow the definition of heterogeneity to the design system level of abstraction. We consider the services provided by PEs, viewing them as computational *"black boxes"*.

Definition 1 : Heterogeneity refers to SoCs that use multiple types of PEs.

As per this definition, heterogeneity becomes a technical characteristic, explicitly describing the processing aspect of the processing, storage, communication triplet defining an SoC. Depending on its class, a PE can be application-specific, generic, programmable, or reconfigurable. All PEs encapsulate processing, storage, and communication logic. This definition leads us to propose the following :

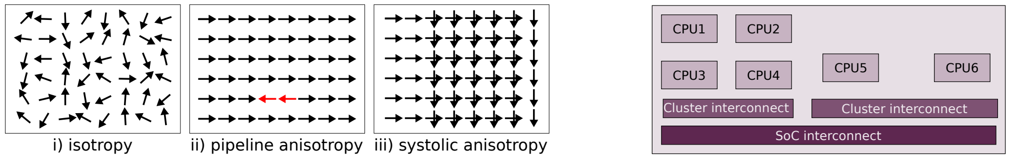
Property 1 : Heterogeneity can refer to both a hardware layout or a Model of Architecture.

A PE's *"class"* or *"type"* can refer to its internal architecture or the service it provides to a processing task. Homogeneity indicates replicating Integrated Circuit (IC) layout sections at a hardware layout level. At an MoA level, heterogeneity refers to the diversity of services provided to the software layers.

Consider a CMOS-built SoC with the architecture depicted in figure 2b. Cores, which we term Core-Level PEs (CL-PEs), share identical internal logic. In this example, clusters can be seen as Cluster-Level PEs (C-PEs), capable of independently retrieving, processing, and sending data back on the SoC interconnect. If we model the architecture per cluster (one C-PE equals one cluster), the architecture is heterogeneous due to different computing capabilities among clusters. Conversely, the architecture is homogeneous if we model per core (one CL-PE equals one core). Both qualifiers (homogeneous and heterogeneous) can apply to the same hardware layout, depending on the model chosen. This example shows that heterogeneity is model-dependent, not solely determined by the system hardware implementation. This paper will focus on MoA heterogeneity, as this aspect is crucial when designing a comprehensive system.

3.2. Anisotropy

We turn our attention to the concepts of isotropy and anisotropy in relation to SoC architectures. Isotropy implies uniformity across all orientations. Anisotropy, on the other hand, describes a



(a) Illustrating SoC anisotropy. (b) Illustrating SoC heterogeneity.

FIGURE 2 – Illustrating SoC anisotropy and heterogeneity.

scenario where properties change according to direction. Applying this to SoC architectures leads us to :

Definition 2 : Anisotropy is a characteristic of SoC architectures where certain pathways or directions for data transfer are more favorable than others, enhancing efficiency in implementation or operation.

An anisotropic SoC architecture makes data transfers along preferred pathways or "directions" more cost-effective in throughput than data sharing along other routes. Anisotropy benefits architectures prioritizing throughput by concentrating on a coarse-grain datapath that separates source (SRC) and sink (SNK) data. Figure 2a provides a visual representation of anisotropy, showing the primary directions for data transfer across the SoC surface.

Property 2 :Anisotropy refers to either a hardware layout or a Model of Architecture.

Regarding hardware layout, data buses that transport information in a single direction, serial or parallel, exhibit anisotropy. Regarding the MoA, consider the same SoC architecture illustrated in Figure 2b. To simplify, we'll use a cluster-level model representing a heterogeneous architecture model with two PEs and one communication medium. At this architectural model level, limiting the data flow direction in the MoA to a singular pathway is feasible. By reducing the options (from two directions to one), the model exhibits anisotropy according to Definition 2. However, the underlying hardware remains isotropic because it allows data to flow in either direction at an equal cost. Considering anisotropy at the model level allows us to view the SoC as a platform providing services to the applications running on it. Here, "services" refer to optimized pathways for data transfer, contributing to improved application performance. Essentially, the SoC is a 'service provider,' akin to a Platform as a Service (PaaS) model in cloud computing, offering a platform that includes the necessary resources (like optimized data transfer pathways) for efficient application execution.

4. Experimental Setup : Building and Testing an Open Source Configurable Anisotropic SoC

RISC-V and open hardware technologies open innovation opportunities in hardware architectures. LiteX provides an infrastructure for SoC prototyping and can be deployed on different hardware platforms. Terasic DE10-Lite board with an Intel MAX10 FPGA has been chosen for the implementation.

We developed a smart camera application (Fig 1b) including a handwritten digit classification Support Vector Classification (SVC) model and an encryption Advanced Encryption Standard (AES) algorithm. The Support Vector Machines (SVMs) model takes a 28×28-pixel image and returns an unsigned char corresponding to the digit written on the image (ranging from 0 to 9). The "MNIST Model" actor represents the SVM model with a linear kernel, trained on the MNIST database [1] using Scikit-learn [10]. The choice of a linear kernel is justified by the trade-off between the number of support vectors and intercepts and the model's accuracy. The generated C code for the model is obtained using the Scikit-porter library [1]. The SDF model of the application includes two actors, FIFO, a SRC storage, and a SNK storage.

The developed system is synchronous and integrates two heterogeneous PEs running at 50MHz as depicted in Figure 4b. One PE has a FemtoRV RISC-V core, while the other has a FireV RISC-V core.Figure 4a presents the internal structure of each PE. The FireV core is faster than the FemtoRV core by 1.66 on SVMs and 1.26 on AES. Three different MoAs are considered for this platform (Fig 3). MoA_i is isotropic, while MoA_{a1} and MoA_{a2} are anisotropic. Most of the 78 other edge direction configurations result in only one active core or invalidate all algorithm-to-architecture mappings. Other anisotropic MoAs allowing some bidirectional communication could be considered but with direction-dependent transfer costs. The PEs communicate via a one-place FIFO on shared memory, and MNIST and AES are forced to be mapped to different cores to exploit pipelining. The predicted digit FIFO is mapped to the shared memory regardless of the mapping. A Wishbone bus connects the PEs and shared memory, with both PEs acting as masters on the bus and a round-robin arbiter implemented in LiteX.

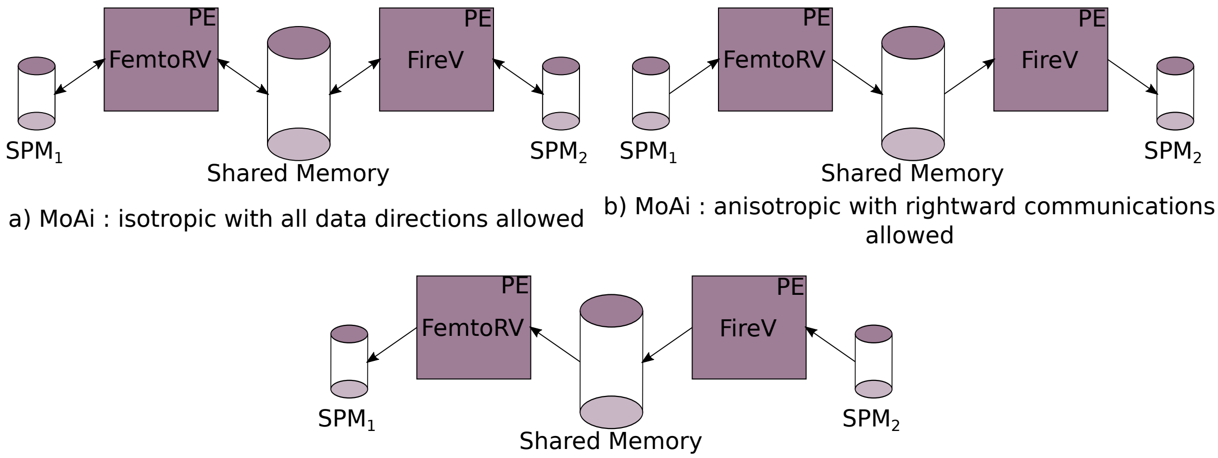
Table 1 lists the valid mappings for different MoAs. Table 2 outlines the memory requirements for each actor. Shared memory has a fixed size of $256\ \mathsf{Bytes}$ except for mappings 9 and 10 ($128\ \mathsf{kBytes}$). We study the effect of operational intensity by simulating data movement with repeated transfers of identical values.

TABLE 1 – Algorithm-to-architecture mappings with supported MoAs. MoA_{a1} and MoA_{a2} are anisotropic while MoA_i is isotropic.

Mapping	MoAs	SPM_1	FemtoRV	Shared Memory	FireV	SPM_2
Mapping 1	MoA_{a1}, MoA_i	Src	MNIST	Pred	AES	Snk
Mapping 2	MoA_i	Src/Snk	MNIST	Pred	AES	-
Mapping 3	MoA_i	Snk	MNIST	Pred	AES	Src
Mapping 4	MoA_i	Src	AES	Pred	MNIST	Snk
Mapping 5	MoA_i	Src/Snk	AES	Pred	MNIST	-
Mapping 6	MoA_i	-	AES	Pred	MNIST	Src/Snk
Mapping 7	MoA_i	-	MNIST	Pred	AES	Snk/Src
Mapping 8	MoA_{a2}, MoA_i	Snk	AES	Pred	MNIST	Src
Mapping 9	MoA_i	-	AES	Pred/Snk/Src	MNIST	-
Mapping 10	MoA_i	-	MNIST	Pred/Snk/Src	AES	-

TABLE 2 – Memory needed for each actor, allocated in the corresponding PE internal memory (power of 2 region size is required).

Actor	Memory Region	Used Size	Region size
MNIST	SRAM 1	45660	64 kB
	SRAM 2	-	-
AES	SRAM 1	28492	32kB
	SRAM 2	5588	8 kB

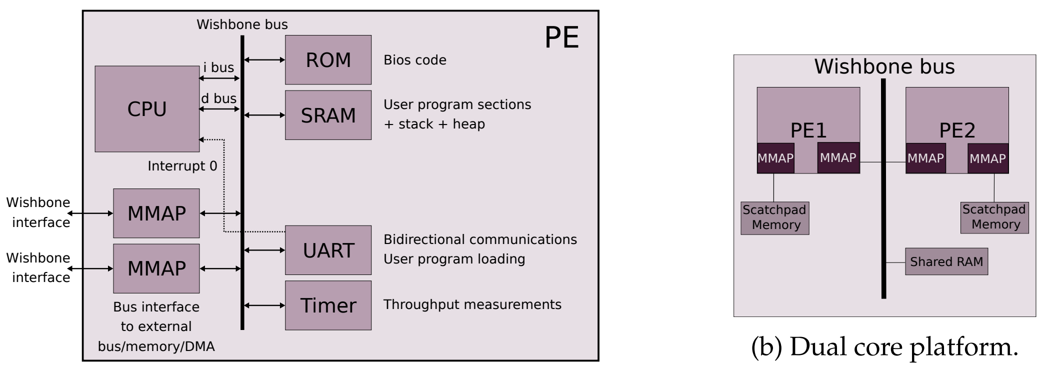


a) MoAi : isotropic with all data directions allowed

b) MoAi : anisotropic with rightward communications allowed

c) MoAi : aniisotropic with leftward directions allowed

FIGURE 3 – The 3 MoAs considered out of 81 ones considering 2 PEs and 3 memories.



(a) Internal structure of a PE.

(b) Dual core platform.

FIGURE 4 – Illustration of the dual-core platform structure and internal description of each PE

5. Experimental Results

Smart camera throughput versus operational intensity is shown in Figure 5. In the left-hand figure, the hardware heterogeneity is optimally used by mapping the *MNIST Model* actor, which is computationally heavy, on the faster core (FireV). The right-hand figure displays equivalent behaviors but with the *MNIST Model* actor mapped to the slower core (FemtoRV). A global speedup of about 1.7$\times$ is observed when properly exploiting heterogeneity. The operational intensity indicates how many cycles are spent for processing per Byte transferred in the system. The two subplots show equivalent behaviors, so we concentrate on the left-hand plot. The mappings 1 and 10 have a throughput approximately constant independently of the variation of operational intensity. This behavior is partly biased because only data circulating between cores have duplicated accesses in our experiments. The mapping 2 corresponds to the case where the SRC and the SNK are mapped on the Scratchpad Memory (SPM) linked to the PE executing the MNIST classification. When operational intensity is low, the throughput is also low because many AES encrypted blocks need to flow back from FireV to SPM_1. The two other mappings of this set, 3 and 7, have the same behavior, except that the throughput is lower

1. https://github.com/nok/sklearn-porter

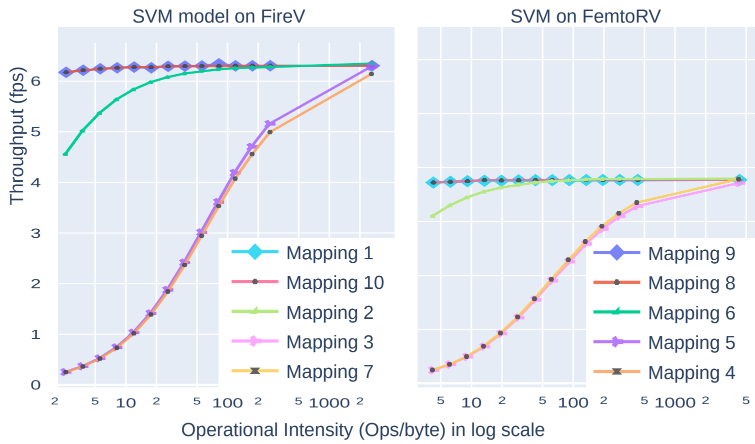


FIGURE 5 – Canonical application throughput versus operational intensity

when the intensity is low because images need to cross Shared Memory. This is explained by the fact that the data flow in the architecture does not follow the natural path of the data, as described in the MoC. Thus, when the amount of data to be transmitted increases, the communication costs increase and add an overhead, thus lowering throughput. The overhead is less critical because the SRC is close to the associated computing core. Mappings 1 and 10 have similar performances because the nature of the chosen application does not foster communication conflicts. As a result, the two mappings are ultimately identical in modeling. However, the fact that the anisotropic MoA selects the best mapping candidate is encouraging future research on anisotropy. Even in this simple example, anisotropy reduces design space exploration complexity, limiting possible mappings to the one efficient for throughput. This result motivates us to conduct further experiments with more complex applications, highlighting communication conflicts and discriminating between solutions such as 1 and 10.

6. Conclusion

This paper has introduced the concept of architecture anisotropy and demonstrated in a small example that using an anisotropic model of architecture can force the system to exploit the natural flow of data, reaching higher throughput and isolating data SRC from data SNK. Anisotropy applies to a hardware layout or an MoA like heterogeneity. Using anisotropic MoAs creates new insight and opportunities for studying whether hardware should be made anisotropic and constrain algorithm-to-architecture mapping solutions to reach higher throughput. We aim in our future works to explore different aspects and forms of anisotropy and to exploit these forms to produce more efficient, highly parallel systems.

Bibliographie

1. Deng (L.). – The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, vol. 29, n6, 2012, pp. 141–142.
2. Dennis (J. B.) et Gao (G. R.). – An efficient pipelined dataflow processor architecture. – In

Supercomputing'88 : Proceedings of the 1988 ACM/IEEE Conference on Supercomputing, Vol. I, pp. 368–373. IEEE, 1988.

3. Fuchs (A.) et Wentzlaff (D.). – The accelerator wall : Limits of chip specialization. – In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–14, Los Alamitos, CA, USA, feb 2019. IEEE Computer Society.
4. Gerbessiotis (A. V.) et Valiant (L. G.). – Direct bulk-synchronous parallel algorithms. *Journal of parallel and distributed computing*, vol. 22, n2, 1994, pp. 251–267.
5. Hennessy (J. L.) et Patterson (D. A.). – A new golden age for computer architecture. *Communications of the ACM*, vol. 62, n2, 2019, pp. 48–60.
6. Johnston (W. M.), Hanna (J. P.) et Millar (R. J.). – Advances in dataflow programming languages. *ACM computing surveys (CSUR)*, vol. 36, n1, 2004, pp. 1–34.
7. Lee (E.) et Messerschmitt (D.). – Synchronous data flow. *Proceedings of the IEEE*, vol. 75, n9, 1987, pp. 1235–1245.
8. Nikhil (R. S.) et al. – Executing a program on the mit tagged-token dataflow architecture. *IEEE Transactions on computers*, vol. 39, n3, 1990, pp. 300–318.
9. Nowatzki (T.), Gangadhar (V.), Ardalani (N.) et Sankaralingam (K.). – Stream-dataflow acceleration. – In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 416–429. IEEE, 2017.
10. Pedregosa (F.), Varoquaux (G.), Gramfort (A.), Michel (V.), Thirion (B.), Grisel (O.), Blondel (M.), Prettenhofer (P.), Weiss (R.), Dubourg (V.), Vanderplas (J.), Passos (A.), Cournapeau (D.), Brucher (M.), Perrot (M.) et Duchesnay (E.). – Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, vol. 12, 2011, pp. 2825–2830.
11. Pelcat (M.). – Models of architecture for dsp systems. *Handbook of Signal Processing Systems*, 2019, pp. 1103–1139.
12. Pelcat (M.), Desnos (K.), Heulot (J.), Guy (C.), Nezan (J.-F.) et Aridhi (S.). – Preesm : A dataflow-based rapid prototyping framework for simplifying multicore dsp programming. – In *Education and Research Conference (EDERC), 2014 6th European Embedded Design in*, pp. 36–40, Sept 2014.
13. Pimentel (A. D.). – Exploring exploration : A tutorial introduction to embedded systems design space exploration. *IEEE Design & Test*, vol. 34, n1, 2016, pp. 77–90.
14. Shen (C.-C.), Wang (L.-H.), Cho (I.), Kim (S.), Won (S.), Plishker (W.) et Bhattacharyya (S. S.). – The dspcad lightweight dataflow environment : Introduction to lide version 0.1. *Technical report UMIACS-TR-2011-17*, 2011.
15. Wolf (W.), Ozer (B.) et Lv (T.). – Smart cameras as embedded systems. *Computer*, vol. 35, 10 2002, pp. 48 – 53.