Models of Architecture: Reproducible Efficiency Evaluation for Signal Processing Systems

Maxime Pelcat, Karol Desnos, Luca Maggiani, Yanzhou Liu, Julien Heulot, Jean François Nezan, Shuvra Bhattacharyya

Models of Architecture: Reproducible Efficiency Evaluation for Signal Processing Systems

Maxime Pelcat*†, Karol Desnos*, Luca Maggiani‡†, Yanzhou Liu§¶, Julien Heulot*, Jean-François Nezan* and Shuvra S. Bhattacharyya§‖

*INSA Rennes, IETR, UBL, Rennes, France
†Institut Pascal, Aubière, France
‡Scuola Superiore Sant'Anna, Pisa, Italy
§University of Maryland, College Park, USA
¶Salzburg University of Applied Sciences, Salzburg, Austria
‖Tampere University of Technology, Tampere, Finland

***Abstract*—The current trend in high performance and embedded signal processing consists of designing increasingly complex heterogeneous hardware architectures with non-uniform communication resources. In order to take hardware and software design decisions, early evaluations of the system non-functional properties are needed. These evaluations of system efficiency require high-level information on both the algorithms and the architecture. In this paper, we define the notion of *Model of Architecture (MoA)* and study the combination of a Model of Computation (MoC) and an MoA to provide a design space exploration environment for the study of the algorithmic and architectural choices. A cost is computed from the mapping of an application, represented by a model conforming a MoC onto an architecture represented by a model conforming an MoA. The cost is composed of a processing-related part and a communication-related part. It is an abstract scalar value to be minimized and can represent any non-functional requirement of a system such as memory, energy, throughput or latency.**

I. Introduction

In the 1990s, models of parallel computation such as the ones surveyed by Maggs et al. in [1] were designed to represent a system including hardware and software-related features. Since the early 2000s, rapid prototyping initiatives such as the Algorithm-Architecture Matching (AAA) methodology [2] have fostered the separation of algorithm and architecture models in order to automate design space exploration.

Models of Computation (MoCs) and especially dataflow MoCs are currently gaining popularity for the design of stream processing systems [3]. Their popularity comes from their capacity to model a parallel application and to guarantee (under certain conditions) functional properties such as deadlock-freeness and memory boundedness while ignoring hardware concerns (instruction set, number of processing elements, etc.).

Modern hardware processing systems are a combination of non-equivalent processing and communication resources, referred to as heterogeneous Multiprocessor Systems-on-Chips (MPSoCs) [4]. The design and programming costs of heterogeneous MPSoCs are constantly rising, because the improvement of programming efficiency is slower than the increase of system complexity. This phenomenon is known as *software and hardware productivity gaps* [4]. In MPSoCs, different sources of heterogeneity arise such as different types of processing units, Non Uniform Memory Access (NUMA) and different types of interprocessor communication (IPC). A unique MoC model is thus unable to represent the properties of an application and the resources of the hardware platform.

In this paper, the notion of Model of Architecture (MoA) is introduced. The main goal of an MoA is to offer standard, reproducible ways to evaluate the efficiency of design decisions. Reproducibility signifies that the model alone, without an associated implementation, is sufficient to reproduce the cost computation. One may note a difference between system performance and system efficiency. In computer science, and considering an application alone, performance is often a synonym of throughput [5][6]. However, signal processing system design requires decisions based on many non-functional costs such as memory, energy, throughput, latency, or area. These costs can be seen as the different modalities of a system's efficiency. In order to evaluate these non-functional costs, an MoA models the internal behavior of an architecture at a high level of abstraction.

Modern streaming applications such as telecommunication, video processing, or deep learning, require a great amount of computation. An early evaluation of the system efficiency is a valuable tool for system designers, as shown by company products such as Poly-Platform from PolyCore Software, Inc., SLX Explorer from Silexica or Pareon from Vector Fabrics whose objectives include providing early performance numbers to the designers. These tools have internal performance models but no standard approach is shared between them. MoAs complement the work on MoCs in providing precise semantics for the second input of the Y-chart [7]. The Y-chart separates the description of an application from the one of an architecture, as illustrated in Figure 1 where algorithm descriptions, conforming to a precise MoC are combined with architecture descriptions conforming to an MoA.

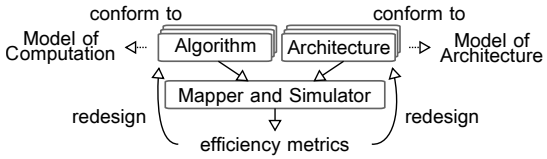


Fig. 1: MoC and MoA in the Y-chart [7].

The paper is organized as follows: Section II presents state of the art MoCs for parallel computation. Section III introduces the notion of MoA and Section IV proposes a new MoA named Linear System-Level Architecture Model (LSLA). Section VI demonstrates systematic, reproducible cost computation from algorithm and architecture models.

II. State of the art of MoCs

The objective of this paper is to sketch the contours of MoAs as the architectural counterparts of MoCs, thereby helping to bridge the increasing gap between MoC-based application design, and platform-based hardware/software implementation. This section introduces a

few representative MoCs that are relevant to signal processing system design. These MoCs will be used in Section VI to demonstrate the proposed LSLA MoA.

Many MoCs have been designed to represent the behavior of a system. The Ptolemy II project [8] has had a considerable influence in promoting MoCs with precise semantics and different forms. Different families of MoCs exist such as finite state machines, process networks, Petri nets, synchronous MoCs and functional MoCs. This paper leverages on both dataflow MoCs and the Bulk Synchronous Parallel (BSP) MoC for their capacity to represent parallel computations, and their relevance to signal processing system design. Section II-A presents static and dynamic dataflow models while Section II-B introduces the BSP MoC.

A. Dataflow MoCs

A dataflow MoC represents a streaming application with a graph where vertices, named actors, represent computation and exchange data through First In, First Out data queues (FIFOs). The unitary exchanged data is called a data token. Computation is triggered when the data present on the input FIFOs of an actor respect its *firing rules*. Dozens of different dataflow MoCs have been explored [9] and this diversity of MoCs demonstrates the benefit of precise semantics and reduced model complexity.

1) The Synchronous Dataflow (SDF) MoC: Synchronous Dataflow (SDF) [10] is the most commonly used dataflow MoC [11]. SDF has a limited expressivity and an extended analyzability. Production and consumption token rates set by firing rules are fixed scalars in an SDF graph and for that reason, SDF is commonly called a static dataflow MoC.

Static analysis can be applied on an SDF graph to determine whether or not fundamental consistency and schedulability properties hold. Such properties, when they are satisfied, ensure that an SDF graph can be implemented with deadlock-free execution and FIFO memory boundedness.

An SDF graph (Figure 2) is defined as $G = \langle A, F \rangle$ where A is the set of actors, and F is the set of FIFOs. For an SDF actor, a fixed integer-valued data rate is set for each port by the function $rate : P^{in}_{data} \cup P^{out}_{data} \rightarrow \mathbb{N}^*$ where P^{in}_{data} is the set of input ports, P^{out}_{data} is the set of output ports for an actor and $\mathbb{N}^*$ is the set of strictly positive integers. A delay $d : F \rightarrow \mathbb{N}$ is set for each FIFO $f \in F$, corresponding to a number of tokens initially present in the FIFO F. $\mathbb{N}$ refers to the set of non negative integers.

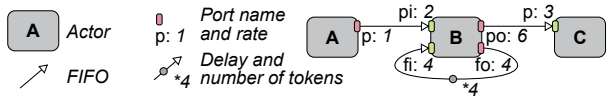


Fig. 2: Example of an SDF Graph.

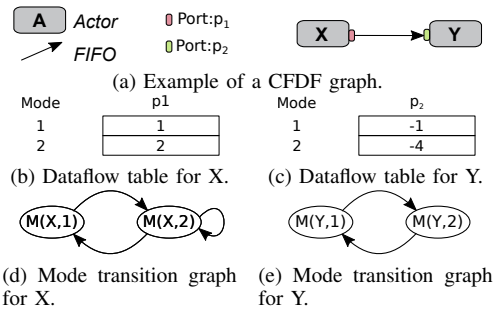
If an SDF graph is consistent and schedulable, a fixed sequence of actor firings, called a *graph iteration*, can be repeated indefinitely to execute the graph, and there is a well defined concept of a minimal sequence for achieving an indefinite execution with bounded memory.

The notion of graph iteration will be used to compute the cost of mapping an SDF algorithm model on an LSLA architecture model in Section VI-A.

2) The Enable-Invoke Dataflow (EIDF) and Core Functional Dataflow (CFDF) MoCs: EIDF is a highly expressive form of dataflow MoC that is useful as a common basis for implementing and analyzing a wide variety of specialized dataflow MoCs [12]. While specialized models such as SDF are useful for exploiting specific characteristics of targeted application domains, the more flexibly-oriented MoC EIDF is useful for interfacing different forms of dataflow and providing tool support that spans heterogeneous systems.

In EIDF, the behavior of an actor is decomposed into a set of actor *modes* such that each actor firing operates according to a given mode. At the end of each actor firing, the actor determines a *next mode set*, which specifies the set of possible modes according to which the next actor firing can execute. The dataflow behavior (production or consumption rate) for each actor port is constant for a given actor mode. However, the dataflow behavior for the same port can differ for different modes of the same actor, which allows for specification of dynamic dataflow behavior. An EIDF graph is defined as $G = \langle A, F \rangle$ and notations used to denotes actors, FIFOs, and data ports are identical to these defined in SDF.

In this paper, we focus on a restricted form of EIDF called *core functional dataflow* (*CFDF*) (Figure 3a). CFDF requires the next mode set that emerges from any firing to contain *exactly one* element [13], ensuring model determinacy. The unique actor mode within the next mode set of a CFDF actor firing is referred to as the *next mode* associated with the firing. Dataflow attributes of a CFDF actor are characterized by a *dataflow table* (Figures 3b and 3c). The rows of the table correspond to the different actor modes, and the columns correspond to the actor ports. Given a CFDF actor A, we denote the dataflow table for A by T_A. If m is a mode of A and p is an input port of A, then $T_A[m][p] = -\kappa(m, p)$, where $\kappa(m, p)$ is the number of tokens consumed from p in mode m. Similarly, if q is an output port of A, then $T_A[m][q] = \rho(m, q)$, where $\rho(m, q)$ is the number of tokens produced onto q in mode m. Mode transition for a CFDF actor is represented by a *mode transition graph* (Figures 3d and 3e). Given a CFDF actor A, the mode transition graph for A, denoted $MTG(A)$ is a directed graph in which the vertices corrspond to the modes of A. The edge set of $MTG(A)$ can be expressed as $\{(x, y) \in V_A \times V_A \mid y \in \mu_A(x)\}$, where V_A represents the set of vertices in $MTG(A)$, and $\mu_A(x)$ is the set of possible next modes for actor x. While production and consumption rates for CFDF actor modes cannot be data-dependent, the next mode can be data-dependent, and thus, $\mu_A(x)$ can have any number of elements up to the number of modes in A.



(a) Example of a CFDF graph.

(b) Dataflow table for X.

(c) Dataflow table for Y.

(d) Mode transition graph for X.

(e) Mode transition graph for Y.

Fig. 3: Dataflow attributes of an example CFDF graph.

The combination of the CFDF MoC and the LSLA MoA to compute an efficiency cost will be discussed in Section VI-B.

B. The Bulk Synchronous Parallel MoC

Another example of a MoC for parallel computation is the Bulk Synchronous Parallel (BSP) [14] MoC. BSP analyzes an application into several phases called supersteps. A BSP computation is composed of a set of components A called agents in this paper to distinguish them from the Processing Elements (PEs) in an MoA. Each agent $\alpha \in A$ has its own memory. An agent α can access the memory of another agent β through a remote access (message)

$r(\alpha, \beta)$ via a so-called *router*. Application execution happens in a series of supersteps indexed by $\sigma \in \mathbb{N}$ and consisting of processing efforts, remote accesses and a global synchronization $s(\sigma)$ (Figure 4).

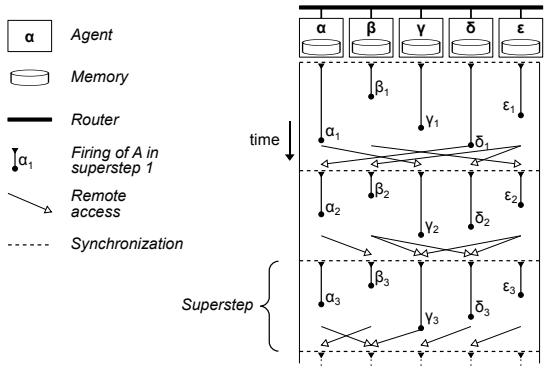


Fig. 4: Example of a BSP Representation.

Each agent α executes in the superstep σ the processing effort α_σ, requiring a time $w(\alpha_\sigma) \in \mathbb{N}$. During the superstep σ, an agent sends or receives at most h_σ remote accesses, each access transferring one atomic data from one agent to another. A barrier synchronization follows each superstep, ensuring global temporal coherency before starting the superstep $\sigma + 1$.

A lower bound for the time of a superstep is computed by:

$$T_\sigma = \max_{0 \leq \alpha < card(A)} w(\alpha_\sigma) + h_\sigma \times g + s \quad (1)$$

where $card(A)$ is the number of agents, g is the time to execute one atomic remote transfer, and s is a fixed synchronization time. A superstep has a discrete length $n \times L$ with $n \in \mathbb{N}$ and L the minimal period of synchronization. Smaller values of L in general lead to superstep times that are closer to the corresponding lower bounds. The BSP execution cost computation is limited to latency and assumes that communication costs for an agent are additive. The combination of the BSP MoC and the LSLA MoA to compute an efficiency cost will be explained in Section VI-C.

Each one of the previous MoCs is characterized by a specific set of properties such as their expressiveness, dynamicity, analyzability, or their decidability. Depending on the complexity and constraints of the modeled application, a simple SDF representation or a more complex CFDF or BSP representation may be chosen. MoCs, by nature, do not carry hardware related information such as resource limitations and hardware efficiency. In this paper, we propose the concept of MoA to complement MoCs in the process of design space exploration.

III. Definition of Models of Architecture (MoAs)

Definition 1. *A **Model of Architecture (MoA)** is an abstract efficiency model of a system architecture that provides a unique, reproducible cost computation, unequivocally assessing a hardware efficiency cost when processing an application described with a specified MoC.*

An MoA does not need to reflect the real hardware architecture of the system. It only aims to represent its efficiency at a coarse grain. As an example, a complete cluster of processors in a many-core architecture may be represented by a single Processing Element (PE) in its MoA representation, hiding the internal structure of this PE. MoAs are intended to be used at a high level of abstraction where hardware, operating system and middleware may be abstracted together. MoAs can be used at all stages of the system design process, from early steps (e.g. to define how many hardware coprocessors are necessary) to late steps (e.g. to optimize runtime scheduling). In addition to classical information such as processor frequency, number of cores or memory architecture, an MoA can give information on physical properties such as energy consumption and temperature dissipation.

In the next section, a new MoA is proposed and named LSLA. This model is providing simple semantics for computing an abstract cost from the mapping of an application described with a precise MoC.

IV. The Linear System-Level Architecture Model (LSLA) MoA

A. Definitions

As an MoA, LSLA provides reproducible cost computation when the activity A of an application is *mapped* on the architecture. Three application-related notions are required prior to the definition of LSLA: application activity, quanta and tokens. These notions are necessary because they make LSLA independent from the chosen MoC.

Definition 2. *The **application activity** A represents the amount of processing and communication necessary for accomplishing the requirements of the application.*

Definition 3. *A **quantum** q is the smallest unit of application activity. There are two types of quanta: processing quantum q_P and communication quantum q_C.*

Two distinct instances of processing quanta are equivalent in the sense that they represent the same amount of computational activity. Processing and communication quanta do not share the same unit of measurement. As an example, in a system with a unique clock and Byte addressable memory, 1 cycle of processing can be chosen as the processing quantum and 1 Byte as the communication quantum.

Definition 4. *A token $\tau \in T_P \cup T_C$ is a non-divisible unit of application activity, composed of a number of quanta. The function $size : T_P \cup T_C \rightarrow \mathbb{N}$ associates to each token the number of quanta composing the token. There are two types of tokens: processing tokens $\tau_P \in T_P$ and communication tokens $\tau_C \in T_C$.*

The activity A of an application is defined by the set:

$$A = \{T_P, T_C\} \quad (2)$$

where $T_P = \{\tau_P^1, \tau_P^2, \tau_P^3 ...\}$ is the set of tokens composing the application processing and $T_C = \{\tau_C^1, \tau_C^2, \tau_C^3 ...\}$ is the set of tokens composing the application communication.

An example of a processing token can be a run-to-completion task. It is composed of N processing quanta (for instance, N cycles). An example of a communication token is a message in a message passing system. It is composed of M communication quanta (for instance M Bytes). Using the two levels of granularity of a token and a quantum, the LSLA MoA can reflect the cost of managing a quantum and the overhead of managing a token composed of several quanta. Definition 5 formally defines the LSLA model, illustrated in Figure 5.

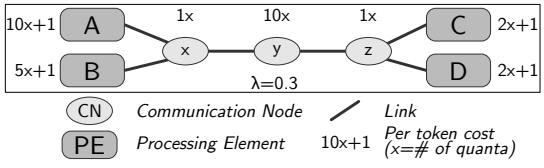


Fig. 5: LSLA MoA semantics elements.

Definition 5. *The Linear System-Level Architecture Model (LSLA) is a Model of Architecture (MoA) that consists of a four-tuple $M = (G, L, cost, \lambda)$. Here, $G = (P, C)$ is an undirected graph where P is the set of abstract architecture Processing Elements (PEs) and C is the set of architecture Communication Nodes (CNs). A processing token τ_P must be mapped to a PE $p \in P$ to be executed. A communication token τ_C must be mapped to a CN $c \in C$ to be transferred. $L = (n_i, n_j), n_i \in C, n_j \in C \cup P$ is a set of undirected links connecting either two CNs or one CN and one PE. A link models the capacity of a CN to communicate tokens to/from a PE or to/from another CN.*

The element of G denoted as "cost" is a function associating a cost to different elements. The cost related to the management of a token τ by a PE or a CN is defined by:

$$\begin{array}{rcccl} cost & : & T_P \cup T_C \times P \cup C & \rightarrow & \mathbb{R} \\ & & \tau, n & \mapsto & \alpha_n.size(\tau) + \beta_n, \\ & & \alpha_n \in \mathbb{R}, \beta_n \in \mathbb{R} & & \end{array} \quad (3)$$

where α_n is the fixed cost of a quantum when executed on n and β_n is the fixed overhead of a token when executed on n. A token communicated between two PEs connected with a chain of CNs $\Gamma = \{x, y, z...\}$ is reproduced $card(\Gamma)$ times and each occurrence of the token is mapped to 1 element of Γ. This procedure is explained on different exemples in Section VI. A token not communicated between two PEs, i.e. internal to one PE, does not cause any cost.

The cost of the execution of application activity A on an LSLA model M is defined as:

$$cost(A, M) = \sum_{\tau \in T_P} cost(\tau, map(\tau)) + \lambda \sum_{\tau \in T_C} cost(\tau, map(\tau)) \quad (4)$$

where $map : T_P \cup T_C \rightarrow P \cup C$ is a surjective function returning the architecture elements where a token is mapped.

$\lambda \in \mathbb{R}$ is a lagrangian coefficient setting the Computation to Communication Cost Ratio (CCCR), i.e. the cost of a single communication quantum relative to the cost of a single processing quantum.

V. Related Work on MoAs

Model	Abstr-action	Distributed Memory	Cost type(s)	Reproducible cost
UML Marte [15]	- -	yes	multiple	no
AADL [16]	-	yes	multiple	no
[4]	-	yes	multiple	no
[17]	+	yes	multiple	yes
[18]	+	yes	time	no
[19]	++	yes	multiple	no
S-LAM [20]	++	yes	time	no
LSLA	+++	yes	abstract	yes

TABLE I: Properties of different state of the art architecture models.

The concept of MoA is evoked in [21] where it is defined as "a formal representation of the operational semantics of networks of functional blocks describing architectures". This definition is very broad, and allows the concepts of MoC and MoA to overlap. As an example, an SDF graph representing a fully specialized system may be considered as a MoC because it formalizes the application. It may also be considered as an MoA because it fully complies with the definition from [21]. This is in contrast to the orthogonalization between MoC and MoA representations that is supported in our proposed modeling framework. Moreover, contrary to the definition from [21], our proposed definition of MoA does not compel the MoA to match the internal structure of the hardware architecture, as long as the generated cost is of interest.

Table I references architecture models of abstract heterogeneous parallel architectures. An evaluation of the level of abstraction of each model is given, as well as some properties.

UML Marte [15] is a system modeling standard offering a holistic approach encompassing all aspects of real-time embedded systems. The standard consists of Unified Modeling Language (UML) classes and stereotypes. As a specification language, UML Marte does not standardize how a cost should be derived from the specified hardware resources and non-functional properties. In contrast, an MoA focuses on reproducible cost computation.

The Architecture Analysis and Design Language (AADL) language [16] defines a syntax and semantics to describe both software and hardware components. The language constructs match logical and physical features such as threads and processes for software and bus and memory for hardware. In contrast, MoAs offer abstract features for describing hardware architectures and delegate responsibility for modeling algorithms to MoCs.

Castrillón and Leupers define in [4] a *quasi-MoA* that divides an architecture into PEs. Each PE has a specific Processor Type (PT) associated to multiple costs and attributes such as context switch time and resource limitations. A graph G of PEs is defined where each edge interconnecting a pair of PEs is associated to a Communication Primitive (CP), i.e. a software application programming interface that is used to communicate among *tasks*. A CP refers to a refers to a Communication Resource (CR) and has its own cost model associating different costs to communication volumes taking into account the number of channels and, the amount of available memory in the module, etc. This model is refered to as a *quasi-MoA* because it does not specify the cost computation procedure from the data provided in the model.

In [17], Kianzad and Bhattacharyya present the CHARMED framework that aims at optimizing multiple system parameters represented in pareto fronts. CHARMED uses an MoA composed of a set of PEs and Communication Resources (CR). Each PE has an attribute vector including processor area, processor price, data memory size, instruction memory size, and power consumption. Each CR also has a vector of attributes including power consumption and transmission speed. This model constitutes, to our knowledge, the only existing MoA as stated by Definition 1. Compared to LSLA, the model in [17] is more complex and does not abstract the computed cost, limiting the model to the defined metrics.

In [18], Grandpierre and Sorel define a graph-based quasi-MoA for message passing and shared memory data transfer simulations of heterogeneous platforms. Memory sizes and bandwidths are taken into account in the model. This model can also be considered as a quasi-MoA because the cost computation procedure is not specified.

In [19], Raffin et. al, describe a quasi-MoA for evaluating the performance of a Coarse Grain Reconfigurable Architectures (CGRAs). The model is customized for a processor type named ROMA and based on a graph representing PEs, memories, a network connecting PEs to memories, and a network interconnecting PEs. The model contains memory sizes, network topologies and data transfer latencies. The objective of the model is to provide early estimations of the necessary resources to execute a dataflow application. Contrary to [19], LSLA abstracts the computed cost and provides a reproducible cost computation procedure.

The System-Level Architecture Model (S-LAM) [20] quasi-MoA focuses on timing properties of a distributed system and defines communication enablers such as Random Access Memory (RAM) and Direct Memory Access (DMA). S-LAM is focused on time modeling and does not provide reproducible cost computation.

Compared to all models except [17], LSLA is the only model that abstracts the type of the computed implementation cost. By its

flexibility and low complexity, LSLA is intended to be used both (a) early in the system design process, when high modeling accuracy is not needed, and (b) post-deployment for runtime management decisions, when the decision processing budget is limited. The next sections illustrate the cost computation provided by LSLA when combined with SDF, CFDF, and BSP MoCs (Section II).

VI. Computing the cost of an application execution on an LSLA architecture

A. Computing the cost of an SDF application execution on an LSLA architecture

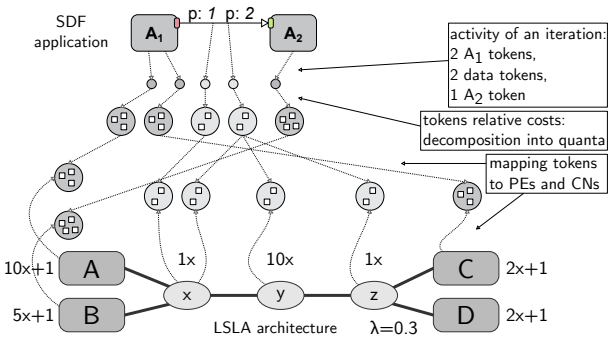


Fig. 6: Computing the cost of executing an SDF graph on an LSLA architecture. The obtained cost for 1 iteration is 31 + 21 + 0.3 (2 + 2 + 20 + 2) + 7 = 66.8 (Equation 4). The cost unit depends on the type of cost represented by the model.

The cost computation mechanism of the LSLA MoA is illustrated by an example in Figure 6 combining an SDF application model and an LSLA architecture model. The scope chosen for the cost computation of a couple (SDF, LSLA), provided that the SDF graph is consistent, is one SDF graph iteration (Section II-A1). Each actor firing is transformed into one processing token and each dataflow token is transformed into one communication token. A token is embedding several quanta (Section IV), allowing a designer to describe heterogeneous tokens to represent firings and data of different weights. In Figure 6, each firing of actor A_1 is associated with a cost of 3 quanta and each firing of actor A_2 is associated to a cost of 4 quanta. Communication tokens represent 2 quanta each.

Each processing token is mapped to one PE (A, B, C or D). Communication tokens are "routed" to the CNs connecting their producer and consumer PEs. For instance, the second communication token in Figure 6 is generating 3 tokens mapped to CNs x, y, and z because the data is carried from PE C to PE B. Since a multiplication by $\lambda = 0.3$ brings the cost of communication tokens to the processing domain, the total cost for communication would be $0.3 \times (2 + 2 + 20 + 2) = 7.8$. The resulting cost from Equation 4 is 66.8. This cost is reproducible and abstract, making LSLA an MoA.

B. Computing the efficiency of a CFDF application execution on an LSLA architecture

Using dynamic dataflow models such as CFDF, a simulation-based integration is a natural way to apply MoA-driven cost computation since there is in general no standard, abstract notion of an application iteration — i.e., no notion that plays a similar role as the periodic schedules of consistent SDF graphs. Figure 7 illustrates an example of execution of a CFDF dataflow graph on an LSLA architecture. We define the cost of execution of actor X in mode $M(X, 1)$ to be 3 quanta and the cost of execution of actor X in mode $M(X, 2)$ to also be 3 quanta. Similarly, the cost of execution of actor Y in mode $M(Y, 1)$ is 2 quanta and the cost of execution of actor Y in mode $M(Y, 2)$ is 4 quanta. These choices represent additional information associated with the CFDF MoC. The cost of communication tokens on the FIFO is set to 2 quanta.

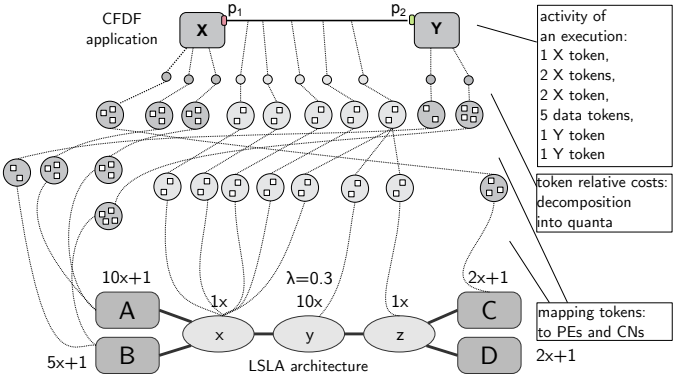


Fig. 7: Computing the cost of executing a CFDF graph on an LSLA architecture. The obtained abstract cost for the chosen simulation scope is $62 + 32 + 0.3\ (10 + 20 + 2) + 7 = 110.6$.

We can then compute a cost for every PE and CN. There are 2 actor tokens mapped to PE A. Each of them has 3 quanta. The cost for PE A is $2 \times (3 \times 10 + 1) = 62$. There are 2 actor tokens mapped to PE B representing 2 and 4 quanta respectively. The cost for PE B is $1 \times (2 \times 5 + 1) + 1 \times (4 \times 5 + 1) = 32$. There is 1 actor token mapped to PE C representing 3 quanta. The cost for PE C is $1 \times (3 \times 2 + 1) = 7$. There are 5 communication tokens mapped to CN x. Each of them has 2 quanta. Therefore, the cost for CN x is $5 \times (2 \times 1) = 10$. Similarly, the cost of y is $1 \times (2 \times 10) = 20$ and the cost of z is $1 \times (2 \times 1) = 2$. The obtained cost is the summation of all PEs' costs and CNs' costs multiplied by λ, which in this example sums up to $62 + 32 + 9.6 + 7 = 110.6$.

C. Computing the efficiency of a BSP application execution on an LSLA architecture

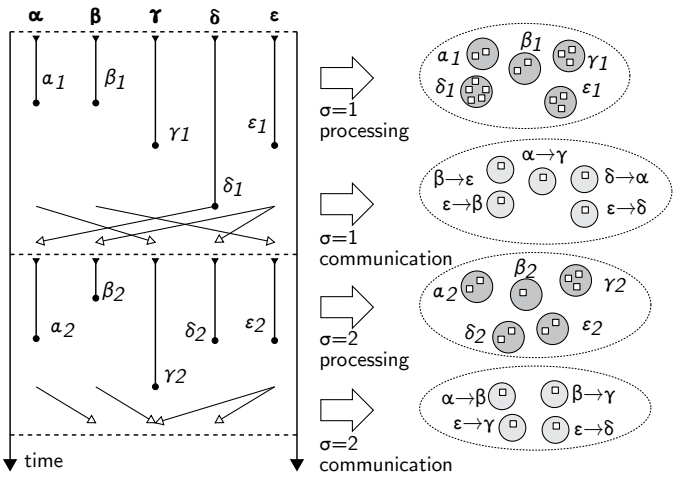


Fig. 8: Extracting the activity of a BSP model.

Figures 8 and 9 illustrate the cost computation of the execution of a BSP algorithm on an LSLA architecture. Figure 8 displays the extraction of the activity from the BSP description, where each processing effort α_σ is transformed into one processing token consisting of $w(\alpha_\sigma)$ quanta (Section II-B) and each (atomic) remote

access is transformed into one communication token of one quantum. Figure 9 shows the mapping and pooling of tokens, consisting of associating tokens to PEs and CNs and replicating communication tokens to route the communications. Agents α and β are mapped on core B, agent γ is mapped on core A, agent ϵ is mapped on core C and agent δ is mapped on core D. The global cost is computed as the sum of the cost of each token on its PE or CN. The communication token $\alpha \rightarrow \beta$ is ignored because it is communicating a token between two agents mapped on the same PE and such a communication has no cost in LSLA, because there is no remote access. The abstract cost

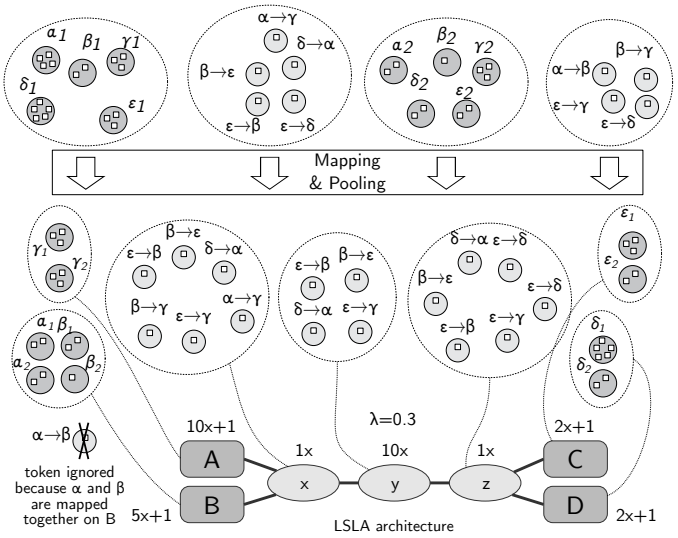


Fig. 9: Computing the cost of executing the BSP model in Figure 8 on an LSLA architecture. The obtained abstract cost is 31 + 31 + 11 + 11 + 11 + 6 + 0.3 (6 + 40 + 6) + 7 + 5 + 11 + 5 = 144.6 (Equation 4).

of 144.6 is obtained for this couple (BSP, LSLA) and, as for SDF and CFDF MoCs, this cost is reproducible as long as the activity extraction from the BSP model follows the same conventions. When compared to using BSP alone, combining BSP and LSLA helps in studying the cost of mapping multiple agents on a single PE — for example, to understand the potential to exploit slack and balance activity between PEs.

D. Discussion on LSLA cost computation

In previous sections, the cost computation mechanisms of LSLA have been demonstrated on static SDF dataflow, dynamic CFDF dataflow and BSP MoCs. The generic and reproducible cost computation of LSLA make LSLA an MoA. While CNs with high cost (such as y in Figure 9) represent *bottlenecks* in the architecture, i.e. communication media with low data rates, PEs with high cost (such as A in Figure 9) represent processing facilities with limited processing efficiency. Each PE can indifferently model a General Purpose Processor (GPP), a Graphics Processing Unit (GPU), or a Digital Signal Processor (DSP) core executing software as well as a hardware component implemented on a Field-Programmable Gate Array (FPGA) or an Application-Specific Integrated Circuit (ASIC).

VII. Conclusion

High performance and embedded signal processing systems require increasingly heterogeneous architectures. Reproducible models are necessary in order to model reliably system-level efficiency. In this paper, the notion of Model of Architecture (MoA) has been defined. An MoA models the behavior of an architecture at a high level of abstraction. When combined to an application model conforming to a given MoC, an architecture model conforming to an MoA provides reproducible cost computation mechanisms for evaluating non-functional system properties such as memory, energy, throughput, etc.

An MoA called Linear System-Level Architecture Model (LSLA) has been introduced and compared to the state of the art of architecture models. LSLA represents hardware efficiency with a linear model, summing the influences of processing and communication on system efficiency.

In future publications, we intend to demonstrate the capabilities of different MoAs to feed efficiency evaluations of systems, optimizing various non-functional properties.

References

[1] B. M. Maggs, L. R. Matheson, and R. E. Tarjan, "Models of parallel computation: A survey and synthesis," in *Proceedings of the HICSS Conference*, vol. 2. IEEE, 1995, pp. 61–70.

[2] T. Grandpierre and Y. Sorel, "From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations," in *Proceedings of the MEMOCODE'03 Conference*. IEEE, 2003, pp. 123–132.

[3] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, *Handbook of signal processing systems*. Springer, 2013.

[4] J. Castrillon Mazo and R. Leupers, *Programming Heterogeneous MPSoCs*. Springer, 2014.

[5] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and linearity: an algebra for discrete event systems*. Wiley Ltd, 1992.

[6] N. Bambha, V. Kianzad, M. Khandelia, and S. S. Bhattacharyya, "Intermediate representations for design automation of multiprocessor DSP systems," *Design Automation for Embedded Systems*, vol. 7, no. 4, pp. 307–323, 2002.

[7] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *Proceedings of ASAP Conference*. IEEE, 1997.

[8] J. Eker, J. W. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, Y. Xiong *et al.*, "Taming heterogeneity-the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[9] H. Yviquel, "From dataflow-based video coding tools to dedicated embedded multi-core platforms," Ph.D. dissertation, Rennes 1, 2013.

[10] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, 1987.

[11] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, 1995.

[12] W. Plishker, N. Sane, M. Kiemb, and S. S. Bhattacharyya, "Heterogeneous design in functional DIF," in *Proceedings of SAMOS VIII*, 2008.

[13] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya, "Functional DIF for rapid prototyping," in *Proceedings of the RSP Symposium*, 2008, pp. 17–23.

[14] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[15] "A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems," 2009.

[16] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (AADL): An introduction," DTIC Document, Tech. Rep., 2006.

[17] V. Kianzad and S. S. Bhattacharyya, "CHARMED: A multi-objective co-synthesis framework for multi-mode embedded systems," in *Proceedings of the ASAP Conference*. IEEE, 2004, pp. 28–40.

[18] T. Grandpierre and Y. Sorel, "Un nouveau modèle générique d'architecture hétérogène pour la méthodologie AAA," *Actes JFAAA*, 2002.

[19] E. Raffin, C. Wolinski, F. Charot, K. Kuchcinski, S. Guyetant, S. Chevobbe, and E. Casseau, "Scheduling, binding and routing system for a run-time reconfigurable operator based multimedia architecture," in *Proceedings of the DASIP Conference*. IEEE, 2010, pp. 168–175.

[20] M. Pelcat, J.-F. Nezan, J. Piat, J. Croizer, and S. Aridhi, "A system-level architecture model for rapid prototyping of heterogeneous multicore embedded systems," in *Proceedings of the DASIP conference*, 2009.

[21] B. Kienhuis, E. F. Deprettere, P. Van Der Wolf, and K. Vissers, "A methodology to design programmable embedded systems," in *Embedded processor design challenges*. Springer, 2002, pp. 18–37.