Models of Architecture

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

Models of Architecture

Technical Report PREESM/2015-12TR01, 2015

Maxime Pelcat[1,3], Karol Desnos[1], Luca Maggiani[2,3], Yanzhou Liu[4],
Julien Heulot[1], Jean-François Nezan[1], and
Shuvra S. Bhattacharyya[4,5]

[1]IETR/INSA Rennes
[2]Scuola Superiore Sant'Anna, Pisa
[3]Institut Pascal, Clermont Ferrand
[4]University of Maryland, College Park
[5]Tampere University of Technology, Tampere

December 15, 2015

Abstract

The current trend in high performance and embedded computing 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 state of the art Model Driven Engineering (MDE) methods, different communities have developed custom architecture models associated to languages of substantial complexity. This fact contrasts with Models of Computation (MoCs) that provide abstract representations of an algorithm behavior as well as tool interoperability.

In this report, we define the notion of *Model of Architecture (MoA)* and study the combination of a MoC and an MoA to provide a design space exploration environment for the study of the algorithmic and architectural choices. An MoA provides reproducible cost computation for evaluating the efficiency of a system. A new MoA called Linear System-Level Architecture Model (LSLA) is introduced and compared to state of the art models. LSLA aims at representing hardware efficiency with a linear model. The computed cost results from the mapping of an application, represented by a model conforming a MoC on 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.

1 Introduction

In the 1990s, models of parallel computation such as the ones over-viewed by Maggs et al in [1] were designed to represent a global 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 is due to the capacity of dataflow MoCs to model a parallel application and to guarantee (under certain conditions) functional properties such as deadlock-freeness and memory boundedness while ignoring hardware concerns (type and instruction set of the processing elements, number of available processing elements, etc...).

Simplifying the design of distributed systems is an important objective due to the widening *software and hardware productivity gaps* [4] that reveal the constantly increasing costs of system design and programming.

Modern hardware processing systems are a combination of non-equivalent processing and communication resources, referred to as heterogeneous Multiprocessor Systems-on-Chips (MPSoCs). The design and programming costs of heterogeneous MPSoCs is constantly rising, because the improvement of programming efficiency is slower than the increase of system complexity. In MPSoCs, different sources of heterogeneity arise such as:

- *Different types of processing units.* As an example, Texas Instruments Keystone II processors [5] embed GPP and DSP cores as well as hardware coprocessors. As another extreme example, the Xilinx Ultrascale MPSoC architecture [6] embeds on a single die microcontrollers, GPPs, GPUs and programmable logic.

- *Non Uniform Memory Access (NUMA).* For instance, Intel Haswell General Purpose Processors (GPPs) have 4 levels of cache.

- *Different types of interprocessor communication (IPC).* As an example, Kalray MPPA many-core processors [7] embed clusters with internal shared memory that communicate with each other through a Network-on-Chip (NoC).

MPSoCs are designed to address a large range of applications, for instance in the telecommunication, military, multimedia and spatial domains. As a consequence, the topologies of these processors and their parallelism differ from the topology and parallelism of the applications they execute. A unique MoC model is thus unable to represent the properties of an application and the resources of the hardware platform.

In this report, the notion of Model of Architecture (MoA) is introduced. A parallel is drawn between dataflow Models of Computation and MoAs. The main goal of an MoA is to offer standard, reproducible ways to evaluate the efficiency of design decisions. 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 [8][9]. However, 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. The MoA provides an unequivocal cost metric computation for a given non-functional cost category.

Modern streaming applications such as telecommunication processing, video processing or deep learning, require an ever greater amount of computation. An early evaluation of the system efficiency is a valuable tool for system designers, as demonstrated by company products such as Poly-Platform from PolyCore Software, Inc. [10], SLX Explorer from Silexica [11] or Pareon from Vector Fabrics [12] whose objectives include providing feedback to the designer some early performance numbers. 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 [13]. 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.

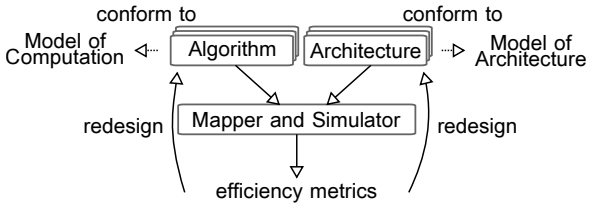


Figure 1: MoC and MoA int the Y-chart [13].

The report is organized as follows: Section 2 presents a static and a dynamic dataflow MoCs, as well as the Bulk Synchronous Parallel (BSP) MoC for parallel computation. These models have inspired the idea of MoA by their simplicity and expressiveness. Section 3 introduces the notion of MoA and Section 4 proposes a new MoA named LSLA. Section 5 presents a discussion on related works and Section 6 demonstrates the cost computation from algorithm and architecture models. Finally, Section 7 concludes the report.

2 State of the art of MoCs

As illustrated in Figure 1, the objective of this report is to sketch the contours of MoAs as the architectural counterparts of MoCs. This section introduces a few MoCs that will be used in Section 6 to demonstrate the cost computing capabilities of the proposed LSLA MoA.

Many Models of Computation have been designed in recent decades to represent with a high level of abstraction the behavior of a system. The Ptolemy II project [14] from the University of California Berkeley has had a considerable influence in promoting MoCs with precise semantics and different forms.

Different families of MoCs exist such as finite state machines, discrete events, process networks, petri nets, synchronous MoCs and functional MoCs [15]. This

report leverages on both dataflow MoCs and the BSP MoC for their capacity to represent a parallel computation. Any MoC which activity can be decomposed into processing and communication tokens (defined in Section 4) can be used with the LSLA MoA proposed in this report. Section 2.1 presents a static and a dynamic dataflow models while Section 2.2 introduces the BSP MoC.

2.1 Dataflow MoCs

Dataflow MoCs constitute an important class of MoCs targeting the modeling of streaming applications. Dozens of different dataflow MoCs have been explored [16] and this diversity of MoCs demonstrates the benefit of precise semantics and reduced model complexity.

A dataflow MoC represents an application behavior 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*. Dataflow MoCs are suitable for representing streaming applications, i.e. applications that process a stream of ordered data.

In order to draw a parallel between Models of Computation and Models of Architecture, two examples of dataflow MoCs, namely SDF and CFDF, are presented in this section and their benefits for system design are discussed.

The Synchronous Dataflow (SDF) MoC

SDF [17] is certainly the most commonly used Dataflow Process Network (DPN) MoC [18]. 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 (assuming that sufficient memory is available on the target platform to satisfy the available memory bounds).

For an SDF actor, a fixed, positive-integer-valued data rate is specified for each port by the function $rate : P_{data}^{in} \cup P_{data}^{out} \rightarrow \mathbb{N}^*$ where P_{data}^{in} is the set of all input ports for an actor and P_{data}^{out} is the set of all output ports for an actor. These rates correspond to the fixed firing rules of an SDF actor, i.e. the data pattern firing actor execution. A delay $d : F \rightarrow \mathbb{N}$ is set for each FIFO in the FIFO set F, corresponding to a number of tokens initially present in the FIFO. The lower represented FIFO in Figure 2 holds for instance 4 tokens of delay.

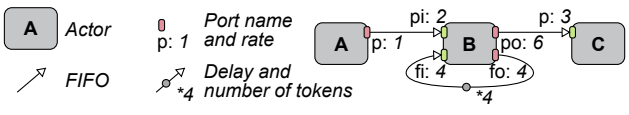


Figure 2: Example of an SDF Graph.

If an SDF graph is consistent and schedulable, a fixed sequence of actor firings 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. Such a minimal sequence is called *graph iteration* and the number of firings of each actor in this sequence is given by the graph Repetition Vector (RV).

Graph consistency means that no FIFO accumulates tokens indefinitely when the graph is executed (preventing FIFO overflow). Consistency can be proven by verifying that the graph topology matrix has a non-zero vector in its null space [17]. When such a vector exists, it gives the RV for the graph. The topology of an SDF graph characterizes actor interconnections as well as token production and consumption rates on each FIFO. A graph is schedulable if and only if it is consistent and has enough initial tokens to execute the first graph iteration (preventing deadlocks by FIFO underflow).

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 6.1. The combination of the SDF MoC and the LSLA MoA to compute an implementation efficiency will be discussed in Section 6.1.

The Enable-Invoke Dataflow (EIDF) and Core Functional Dataflow (CFDF) MoCs

EIDF is a highly expressive form of DPN that is useful as a common basis for representing, implementing, and analyzing a wide variety of specialized dataflow MoCs [19, 20]. While such specialized models, such as SDF [17] (Section 2.1) and Cyclo-Static Dataflow (CSDF) [21], are useful for exploiting specific characteristics of targeted application domains (e.g., see [22]), the more flexibly-oriented MoC EIDF is useful for integrating and interfacing different forms of dataflow, and providing tool support that spans heterogeneous application areas, subsystems, or target platforms. At the same time, EIDF is a generalization of SDF and CSDF, so that techniques for these popular *decidable dataflow models* (e.g., see [23]) can be applied seamlessly within an EIDF framework.

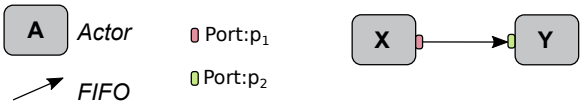
In EIDF, the behavior of an actor is decomposed into a set of mutually exclusive actor *modes* such that each actor firing operates according to a given mode, and at the end of each actor firing, the actor determines a set called the *next mode set*, which specifies the set of possible modes according 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 be different for different modes of the same actor, which allows for specification of dynamic dataflow behavior. EIDF can be qualified as a dynamic dataflow MoC.

An EIDF graph $G = (A, F)$ contains a set of actors A that are interconnected by a set of FIFOs F. An actor $a \in A$ comprises a set of data ports $(P^{in}_{data}, P^{out}_{data})$ where P^{in}_{data} and P^{out}_{data} respectively refer to a set of data input and output ports, used as anchors for FIFO connections. Functions $src : F \rightarrow P^{out}_{data}$ and $snk : F \rightarrow P^{in}_{data}$ associate source and sink ports to a given FIFO.

In this report, we focus on a restricted form of EIDF that is particularly useful in the modeling of signal processing systems. This restricted form, called *core functional dataflow* (*CFDF*), requires that the next mode set that emerges from any actor firing contain *exactly one* element [19]. This restriction is important in ensuring determinacy. The unique element (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 can be characterized by a CFDF *dataflow table*. The rows of the dataflow table correspond to the different actor modes, and the columns correspond to the different actor ports. Thus, an actor with n_m modes and n_p ports has a dataflow table with dimensions $n_m \times n_p$. 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 $\kappa(m, p)$ denotes 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 $\rho(m, q)$ represents the number of tokens produced onto q in mode m. Recall from the definition of EIDF, that $\kappa(m, p)$ and $\rho(m, q)$ are constant values (independent of the data values that are consumed or produced during execution of the mode).



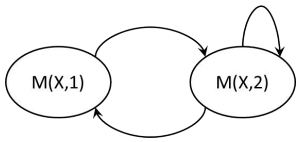
(a) Example of a CFDF graph.

Mode	p_1
1	1
2	2

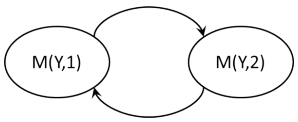
(b) Dataflow table for actor X.

Mode	p_2
1	-1
2	-4

(c) Dataflow table for actor Y.



(d) Mode transition graph for actor X.



(e) Mode transition graph for actor Y.

Figure 3: Dataflow attributes of CFDF.

Figure 3a shows a simple example of a CFDF graph with two actors X and Y. Examples of dataflow tables for these actors are illustrated in Figure 3b and Figure 3c, respectively.

Mode transition behavior for a CFDF actor can be graphically represented by a *mode transition graph*. Given a CFDF actor A, the mode transition graph for A, denoted $MTG(A)$ is a directed graph in which the vertices are in one-to-one correspondence with 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. Note that while production and consumption rates for CFDF actor modes cannot be data-dependent, the next mode can be data-dependent, and therefore, $\mu_A(x)$ can in general have any positive number of elements up to the total number of modes in A.

Figure 3d and Figure 3e illustrate examples of mode transition graphs for the actors X and Y in the CFDF graph example of Figure 3a, respectively. The notation $M(a, b)$ in these figures represents the mode labeled b for the actor a.

For example, from Figure 3d, we see that $\mu_X(1) = \{2\}$, and $\mu_X(2) = \{1, 2\}$.

The developments of this report can be extended to incorporate concepts of *parameterized sets of modes* (*PSMs*), which facilitate the efficient management of related groups of modes in a given CFDF graph [24]. Developing such extensions is a useful direction for future work. The combination of the CFDF MoC and the LSLA MoA to compute an implementation efficiency will be discussed in Section 6.2.

2.2 The Bulk Synchronous Parallel MoC

Another example of a MoC for parallel computation is the Bulk Synchronous Parallel (BSP) [25] MoC. BSP analyzes an application into several phases called supersteps. A BSP computation is composed of a set of components A — we will call them agents in this report 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*. The computation execution happens in a series of supersteps indexed by $\sigma \in \mathbb{N}$ and consisting of processing efforts, remote accesses and a global synchronization $s(\sigma)$. An example of a BSP algorithm model is illustrated in Figure 4.

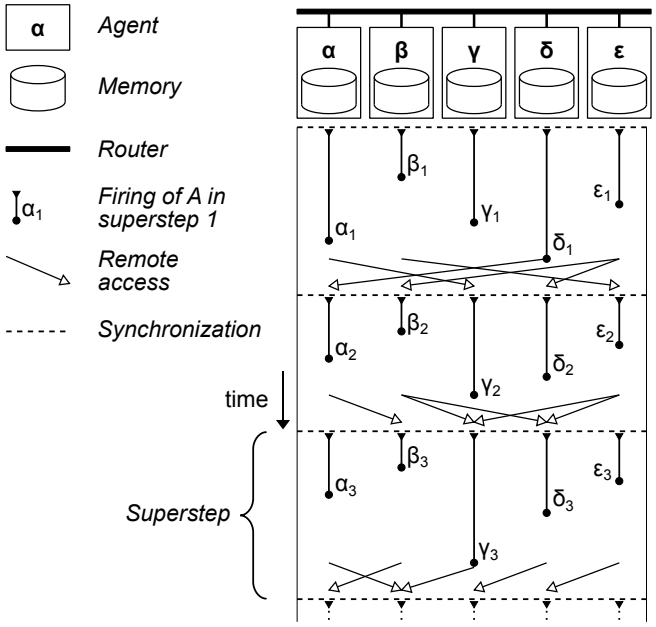


Figure 4: Example of a BSP Representation.

Each agent α executes the processing effort α_σ during the superstep σ. The processing effort α_σ requires a time $w(\alpha_\sigma) \in \mathbb{N}$ to be processed. 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 next superstep $\sigma + 1$.

BSP provides a time performance evaluation for a superstep. A lower bound for the time of a BSP 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, $w(\alpha_\sigma)$ is the time of the processing effort of agent α during superstep σ, h_σ is the maximum number of remote transfers sent and received by a single agent in superstep σ, g is the time to execute one atomic remote transfer, and s is a fixed time cost associated to the synchronization. A superstep has a discrete length $n \times L$ with $n \in \mathbb{N}$ and L the minimal period of synchronization. The smaller L is chosen, the closer from the lower bound T_σ the superstep time results.

This cost computation is limited to latency computation and assumes that communication costs for an agent are additive and that synchronization has a fixed cost. The combination of the BSP MoC and the LSLA MoA will be explained in Section 6.3. Combining the BSP MoC with different MoAs opens new opportunities in terms of agent merging and efficiency computation when compared to using BSP alone.

2.3 Benefits Offered by MoCs

In a CFDF dynamic representation, a conditional *if* statement can be represented by using a parameterized mode and executing the statement in the corresponding actor firing. In a static SDF representation, a conditional statement cannot be represented at graph level and the amount of data flowing between actors is constant over time. In a BSP representation, inter-core synchronization been designed to scale up to thousands or millions of processing cores. Depending on the complexity and constraints of the modeled application, a simple SDF representation or a more complex EIDF or BSP representation can be chosen.

MoCs offer abstract representations of applications at different levels of abstraction. They can be used for early system studies or system functional verification. MoCs simplify the study of a system and, since they do not depend on a particular syntax, they offer advanced interoperability to the tools manipulating them.

MoCs, by nature, do not carry hardware related information such as resource limitations and hardware efficiency. In this report, we propose the concept of MoA to complement MoCs in the process of design space exploration.

3 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.*

As explained by Box and Draper [26], "all models are wrong, but some are useful". 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. As another example, the resources offered by a partially loaded PE can be represented in an MoA by a lower performing processing element.

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. MoAs are intended for Cyber-Physical Systems, where physical properties and processing properties are intricately linked.

An MoA can take different forms. It may be a matrix specifying communication speed between each pair of cores in a system, a vector specifying relative processing capabilities of the different cores or a graph specifying inter-core connectivity.

In the next section, a new MoA is proposed and named LSLA. This model is providing minimal semantics for computing an abstract cost from the mapping of an application described with a precise (and not necessarily dataflow) MoC.

4 The Linear System-Level Architecture Model (LSLA) MoA

In this section, a novel MoA is proposed. As an MoA, LSLA provides reproducible cost computation when the activity A of an application is *mapped* on the architecture.

4.1 Definitions

Three application-related notions need to be defined prior to the definition of LSLA: application activity, tokens and quanta. These notions are necessary because they make LSLA independent from the chosen MoC.

Definition 2. *The application activity A corresponds to the amount of processing and communication necessary for accomplishing the requirements of the application. The application activity is composed of processing and communication tokens.*

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 processing quanta are equivalent, thus represent the same amount of 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*

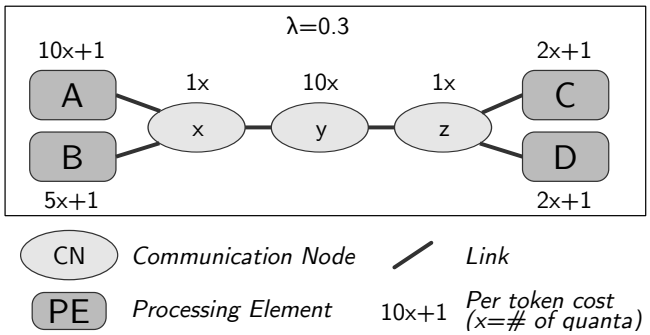


Figure 5: LSLA MoA semantics elements.

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 formally as the set:

$$A = \{T_P, T_C\} \tag{2}$$

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

The LSLA Model of Architecture (MoA) is defined hereafter. The model composing elements are illustrated in Figure 5. An LSLA is composed of Processing Elements, Communication nodes and links. The cost of executing a token on a Processing Element or a Communication Node is affine wrt. the number of quanta in this token. Definition 5 formally defines the LSLA model.

Definition 5. *The Linear System-Level Architecture Model (LSLA) is a Model of Architecture (MoA) that consists of an undirected graph $G = (P, C, L, cost, \lambda)$ where:*

- *P is the set of the architecture Processing Elements (PEs). A PE is an abstract processing facility. A processing token t_P must be mapped to a PE $p \in P$ to be executed.*
- *C is the set of the architecture Communication Nodes (CNs). A communication token t_C must be mapped to a CN $c \in C$ to be executed.*
- *$L = (n_i, n_j), n_i \in C, n_j \in C \cup P$ 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.*

- *cost is a function associating a cost to different elements in the model.*

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 6.

The cost of the execution of application activity A on an LSLA graph G is defined as:

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

where $map : T_P \cup T_C \rightarrow P \cup C$ is a surjective function returning the mapping of each token on one of the architecture elements.

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

5 Related Work

5.1 Related Work on Virtual Platforms

Virtual platforms such as QEMU [27], gem5 [28] or Open Virtual Platforms simulator (OVPsim) have been created as functional emulators that aim to validate software when silicon is not available. The high complexity of these virtual platforms limits the maximum complexity of the emulated hardware. The steadily increasing complexity of MPSoCs motivates the creation of a new level of architecture modeling abstraction and a new type of virtual streaming platforms, focusing on hardware resources and parallelism rather than on cycle accuracy. MoAs are created to fill this gap. MoAs are abstract models for system efficiency evaluation while virtual platforms are linked to hardware features such as instruction sets. Moreover, an MoA can be used to represent the behavior of both software and hardware, in contrast with existing methods that either focus on hardware description or on software coding.

Some initiatives such as UML MARTE [29], SystemC Transaction-level modeling (TLM) [30], AADL [31] or SHIM [32] aim at defining architecture descriptions at a higher level of abstraction than virtual platforms. These standards are focused of global system modeling and do not foster separation of concerns between algorithm-related performance, architecture resources and system requirements. The semantics of different architecture models from the literature are detailed in the next Section and their belonging to the domain of MoAs is discussed.

5.2 Related Works on Architecture Modeling

Many publications are related to architecture modeling. Table 1 references some models and properties. All the presented models abstract heterogeneous parallel architectures and are algorithm-agnostic. A general idea of the level of abstraction of each model is given, as well as some properties.

MoA	Abstr-action	Distributed Memory	Obj-ectives	Reproduc-ible cost
HVP [33]	- - -	no	time	no
UML Marte [34]	- -	yes	multiple	no
AADL [31]	-	yes	multiple	no
[4]	-	yes	multiple	no
[35]	+	yes	multiple	yes
[36]	+	yes	time	no
[37]	++	yes	multiple	no
S-LAM [38]	++	yes	time	no
LSLA	+++	yes	abstract	yes

Table 1: Properties of different state of the art architecture models.

In multi-core rapid prototyping tools, each community is using a different custom architecture model, often associated to a specific syntax and specific requirements. For example, some custom models for multicore scheduling have been proposed such as the High-level Virtual Platform (HVP) [33], the SynDEx architecture model [2], and the System-Level Architecture Model (S-LAM) [38]. These models are time-oriented models for automated multicore scheduling.

HVP [33] defines an evolutionary virtual platform based on SystemC rather than a strict MoA. HVP is presented here to distinguish virtual platforms from MoAs. The corresponding MoC that can be coexplored by HVP is the *Communicating Processes* one [39]. The HVP platform virtually executes tasks and HVP defines task automata for managing the internal behaviour of application tasks over time. HVP targets Symmetric Multiprocessing (SMP) processor architectures. HVP is focused on time simulation and does not provide other types of performance information.

UML Marte [34] is a system modeling standard offering a holistic approach encompassing all aspects of real-time embedded systems. The standard consists in 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 amount of hardware resources and non-functional properties. MoAs focus on abstract cost computation and can be used in the context of UML Marte. The data specified in a UML Marte representation of a system can fill an MoA for computing the cost of a design decision.

The Architecture Analysis and Design Language (AADL) language [31], specified by the SAE International organization, defines a syntax and semantics to describe both software and hardware components in a system. The efforts are focused on building a description language more than in providing a simple model. The language constructs match logical and physical features of the described system such as threads and processes for software and bus and memory for hardware. In contrast to this approach, 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 archi-

tecture into PEs. Each PE has a specific Processor Type (PT) defined by $PT = (CM^{PT}, X^{PT}, V^{PT})$ where CM^{PT} is a set of functions associating costs to PEs, X^{PT} is a set of PE attributes such as context switch time of resource limitations, and V^{PT} is a set of variables such as the processor scheduling policy. A graph G of PEs is defined where each edge interconnecting a pair of PEs is associated to a Communication Primitive (CP). A CP is a software application programming interface that is used to communicate among *tasks*. A CP has its own cost model CM^{CP} associating different costs to communication volumes. It also refers to a Communication Resource (CR), i.e. a hardware module that is used to implement the communication and has information on the number of channels and on the amount of available memory in the module. This model can be refered to as a *quasi-MoA* because it does not specify any cost computation rule from the data provided in the model.

In [36], 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. The model is composed of an oriented graph $G_h = (V, E)$ where E is a set of oriented edges with no property and each vertex $v \in V$ is a finite state machine with one of six types:

- an *operator* is a processing element,
- a Random Access Memory (RAM) is a memory that may be shared between operators. It can be a data, program or mixed memory,
- a Sequential Access Memory (SAM) is a FIFO queue supporting message passing. It can natively support multipoint and broadcast data transfers,
- a *communicator* is equivalent to a Direct Memory Access (DMA) that drives the communication between two operators by accessing RAM or SAM memories,
- a bus/multiplexer/demultiplexer (BMD) without arbiter is a bloc representing together the data multiplexer preparing memory data for bus transfer, the bus itself, and the demultiplexer preparing data for memory access,
- a BMD with arbiter is a BMD with a function, specifying data conflict management when accessing memories.

More details on this architecture model are given (in french) in [40]. This model can also be considered as a quasi-MoA because cost computation is not specified. It is moreover focused on time modeling.

In [35], Kianzad and Bhattacharyya present the CHARMED co-synthesis framework and its MoA. The CHARMED framework aims at optimizing multiple system parameters represented in pareto fronts. The MoA is composed of a set of PEs and Communication Resources (CR). Each PE has a vector of attributes $PE_{attr} = [\alpha, \kappa, \mu_d, \mu_i, \rho_{idle}]^T$ where α denotes the area of the processor, κ denotes the price of the processor, μ_d denotes the size of data memory, μi denotes the instruction memory size and ρ_{idle} denotes the idle power consumption of the processor. Each CR also has an attribute vector: $CR_{attr} = [\rho, \rho_{idle}, \theta]^T$ where ρ denotes the average power consumption per each unit of data to be

transferred, ρ_{idle} idle denotes idle power consumption and θ denotes the worst case transmission rate or speed per each unit of data. This model constitutes, to our knowledge, the only existing MoA as stated by Definition 1. This MoA targets several costs and proposes a pareto-based method. Compared to the LSLA model proposed in this paper, the model in [35] is much more complex and does not abstract the computed cost, limiting the model to the few metrics defined by the authors.

In [37], Raffin et. al, describe a quasi-MoA for evaluating the performance of a Coarse Grain Reconfigurable Architectures (CGRAs) implementation in order to perform constraint programming optimization. The model is customized for the ROMA processor [37] and based on a graph representing PEs (called operators), memories, a network connecting PEs to memories (for local and shared memory accesses), and a network interconnecting PEs (for message passing). The model contains memory sizes, network topologies and data transfer latency information. The objective of the model is to provide early estimations of the necessary resources to execute a dataflow application. Compared to LSLA, the model in [37] does not abstract the computed cost and does not provide a reproducible cost computation procedure.

The S-LAM quasi-MoA focuses on timing properties of a distributed system and additionally defines communication enablers such as RAM and DMA. S-LAM has been used so far for the dataflow design of a video decoder within the MPEG Reconfigurable Video Coding (RVC) framework [41] as well as for the rapid prototyping of a 3GPP Long Term Evolution (LTE) base station [15] and for the design space exploration of a computer vision algorithm [42]. S-LAM is focused on time modeling and does not provide a reproducible cost computation procedure.

In Table 1, many architecture description languages have been omitted because they operate at a different level of abstraction than MoAs. For instance, behavioral VHDL is a language to model a hardware behavior but its extreme versatility does not orientate the designer towards a specific Model of Architecture.

In the next section, the cost computation of LSLA is demonstrated on different MoCs, showing the genericity of the model.

6 Computing the cost of an application execution on an LSLA architecture

A useful approach to applying the cost computation models introduced in this report is to integrate them with functional simulation to provide estimates of performance based on underlying MoAs. This kind of simulation would be faster to execute and easier to adapt compared to a cycle-accurate, timed simulation, while providing some insight about efficiency and application-architecture affinity (in contrast to a purely functional simulation).

The next sections illustrate the cost computation provided by LSLA when combined with SDF, CFDF, and BSP MoCs introduced in Section 2. This combination follows the Y-chart displayed in Figure 1. Different Models of Computation are combined with LSLA and show the genericity of the LSLA MoA.

6.1 Computing the cost of an SDF application execution on an LSLA architecture

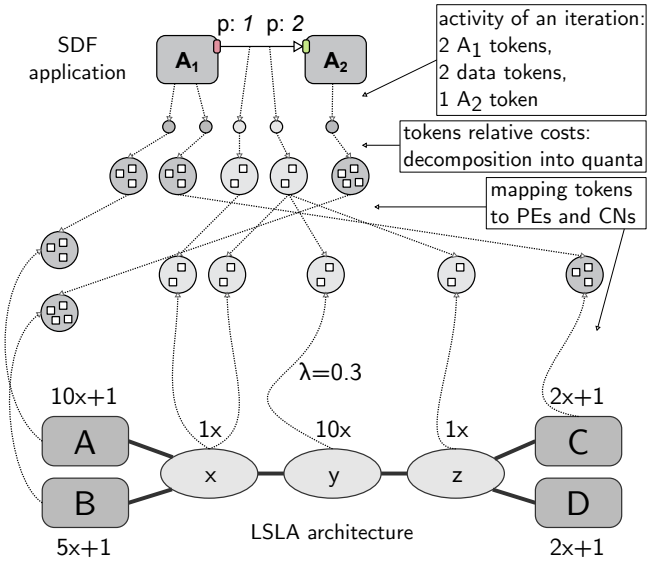


Figure 6: Computing the cost of executing an SDF graph on an LSLA architecture. The obtained cost for 1 iteration is 31 + 21 + 0.3 (2 + 2 + 20 + 2) + 7 = 66.8 (Equation 4).

The cost computation mechanism of Linear System-Level Architecture Model (LSLA) is illustrated by an example in Figure 6 combining an SDF application model and an LSLA architecture model. The cost of the execution of application activity A on LSLA graph G is arbitrarily conducted in the processing domain, as stated in Equation 4. The multiplication by λ brings the cost of communication tokens into the processing domain.

Each actor firing during the studied graph iteration is transformed into one processing token. Each dataflow token transmitted during one iteration is transformed into a communication token. A token is embedding several quanta (Section 4), allowing a designer to describe heterogeneous tokens to represent firings and data of different sizes. 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. The natural scope for the cost computation of a couple (SDF, LSLA), provided that the SDF graph is consistent, is one SDF graph iteration (Section 2.1).

Each processing token is mapped to one PE. 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 x, y, and z because the data is carried from C to B. The resulting cost from Equation 4 is 66.8. This cost is reproducible and abstract, making LSLA an MoA.

6.2 Computing the efficiency of a CFDF execution on an LSLA architecture

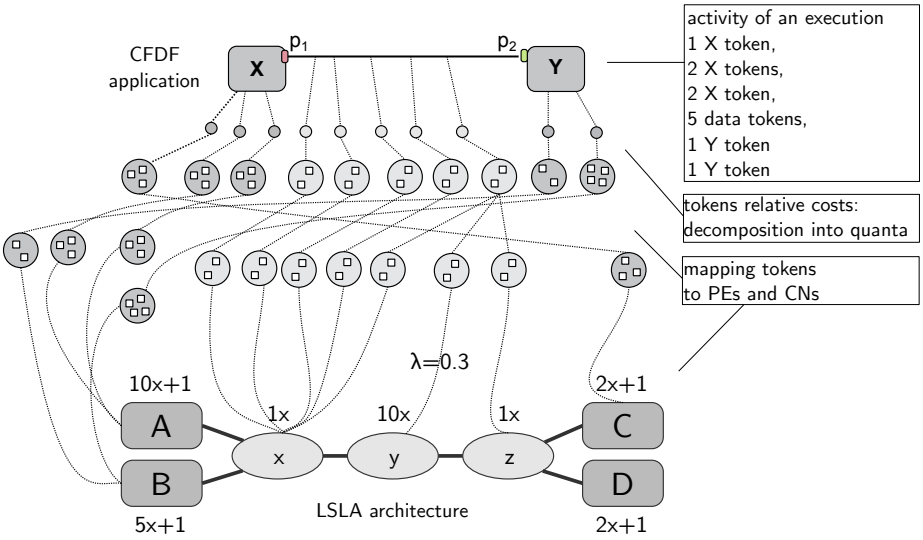


Figure 7: Computing the cost of executing a CFDF graph on an LSLA architecture. The obtained cost for the chosen simulation scope is $62+32+9.6+7 = 110.6$.

For 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 (and their associated repetitions vectors) of consistent SDF graphs.

A simulation-based integration approach may be useful even for SDF graphs because even though the production and consumption rates are statically fixed in SDF graphs (leading to the notion of periodic schedules), the internal behavior of the actors (the sequences of operations executed within the actors) can vary significantly based on the data that is consumed. Models that capture or estimate this variation in terms of input data values can be integrated with functional simulation to provide potential for more accurate performance assessment compared to analysis that is based only on a single iteration of the periodic schedule.

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. 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. They represent 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. It represents 3 quanta. The cost for PE C is $1 \times (3 \times 2 + 1) = 7$. There is no actor token mapped to PE D. Therefore, the cost for PE D is 0. There are 5 communication tokens mapped to CN x. Each of them has 2 quanta. Therefore, the cost for CN x is $5 \times (2 \times 1) = 10$. There is 1 data token mapped to CN y. It has 2 quanta. Therefore, the cost for CN y is $1 \times (2 \times 10) = 20$. There is 1 data token mapped to CN z. It has 2 quanta. Therefore, the cost of z is $1 \times (2 \times 1) = 2$. Since a multiplication by $\lambda = 0.3$ brings the cost of communication tokens to the processing domain, the total cost for communication would be $0.3 \times (10 + 20 + 2) = 9.6$. Therefore, the obtained cost is the summation of all PEs' cost and CNs' cost, which in this example sums up to $62 + 32 + 9.6 + 7 = 110.6$.

6.3 Computing the efficiency of a BSP execution on an LSLA architecture

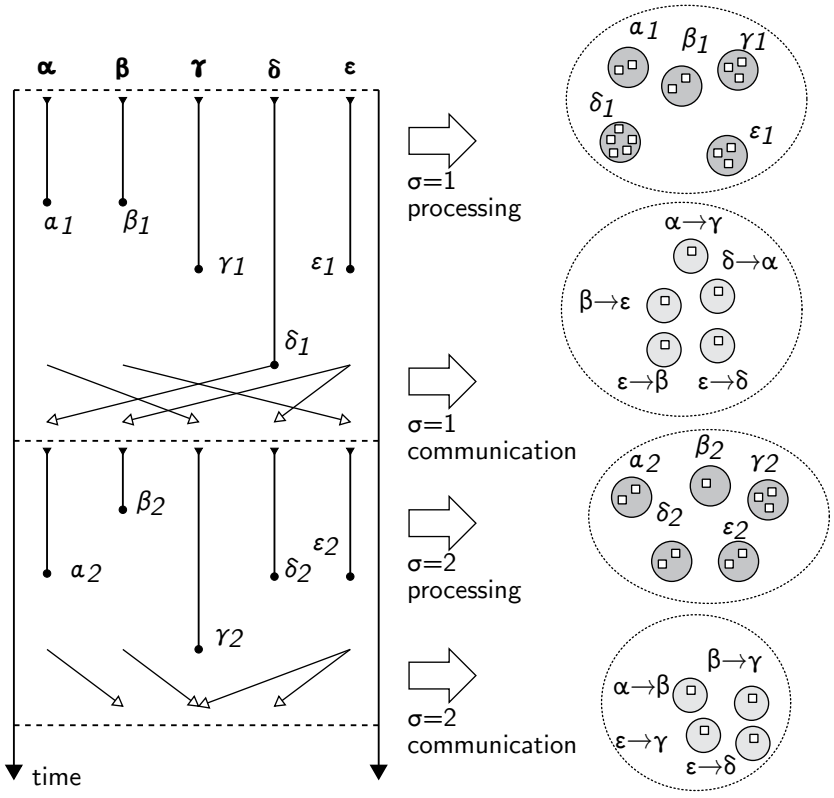


Figure 8: Extracting the activity of a BSP model.

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, consisting of processing and communication tokens, from the BSP description. Each processing effort α_σ is transformed into one processing token

consisting of $w(\alpha_\sigma)$ quanta (Section 2.2) and each remote access is transformed into one communication token of one quantum. This size of one quantum is chosen because the BSP model considers atomic remote accesses.

Figure 9 shows the mapping and pooling of the tokens, consisting on 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 is supposed to have no specific cost, because there is no remote access.

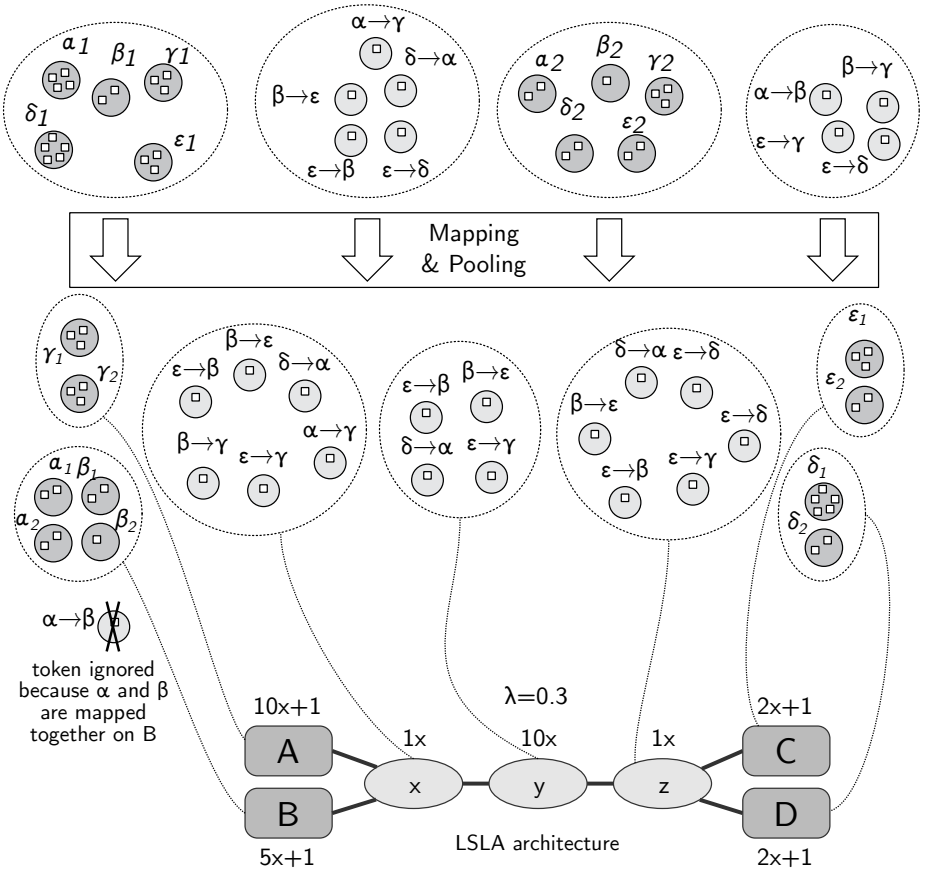


Figure 9: Computing the cost of executing the BSP model in Figure 8 on an LSLA architecture. The obtained cost is 31 + 31 + 11 + 11 + 11 + 6 + 0.3 (6 + 40 + 6) + 7 + 5 + 11 + 5 = 144.6 (Equation 4).

An abstract cost 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 studying the cost of mapping several agents on a single PE, exploiting parallel slack-

ness to balance activity between PEs. Parallel slackness refers to the ability for processing time to dominate communication and synchronization time. One may note that the supersteps with the same σ of the different agents mapped to the same PE are merged.

In the 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 a Model of Architecture. Next section discusses the meaning of the cost expressions in LSLA and compares LSLA to the state of the art models.

6.4 Discussion on LSLA cost computation

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.

For example, the LSLA model at the bottom of Figure 9 may represent a set of two processors $\{A, B\}$ and $\{C, D\}$ where $\{A, B\}$ has a core A and a coprocessor B. B is almost twice as efficient as A (cost of $5x + 1$ instead of $10x + 1$ for each token). $\{C, D\}$ may be a homogeneous bi-core processor with high efficiency (cost of 2 for each firing). $\{A, B\}$ and $\{C, D\}$ are communicating through a link (for instance a serial link) that has one tenth of the efficiency of internal $\{A, B\}$ and $\{C, D\}$ communications (cost of 10 instead of 1 for each token).

Each PE can indifferently model a 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).

The cost computed by LSLA and resulting from communication and processing is linear with respect to the number of communication and processing tokens (Equation 4). This cost can represent either a processing time, a diminution in the throughput, an area, a price, an energy, a power, an amount of memory, etc..., depending on the purpose of the architecture model. Several LSLA models can be combined to represent several non-functional properties of a single system.

Compared to the model in [4], the LSLA model differs in the sense that PEs are not directly linked by edges but rather through communication nodes that can be chained to represent complex PE interconnects. Moreover, LSLA is simpler than the model in [4] and the cost of the internal communication in a PE is taken into account in [4] while it is not taken into account in LSLA.

In [35], all different costs are represented in a single model whereas LSLA defines a single abstract cost model and several models of the same hardware can be combined for multi-objective optimization.

Compared to all other models presented in Section 5.2, the LSLA model is the only one that abstracts the type of the computed implementation cost. By its simplicity, LSLA is intended to be used both early in the system design process, when high modeling accuracy is not needed, and for runtime management decisions, when decision processing budget is limited.

7 Conclusion

In this report, the notion of Model of Architecture (MoA) has been defined. An MoA models the internal 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.

This report inspires from MoCs for parallel computation. An application model, conforming a MoC, is an abstract representation of the application behavior and, by nature, does not give information on the hardware efficiency. When combined with a model of the architecture conforming a precise MoA, the efficiency of the system can be evaluated and the design space explored. Application and architecture can then be modified independently and the cost of these modifications can be analyzed. This efficiency evaluation can serve for many studies such as the optimization of memory, energy, throughput, or latency.

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 performance 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.

Bibliography

[1] Bruce M. Maggs, Lesley R. Matheson, and Robert Endre Tarjan, "Models of parallel computation: A survey and synthesis," in *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*. 1995, vol. 2, pp. 61–70, IEEE.

[2] Thierry Grandpierre and Yves Sorel, "From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations," in *Formal Methods and Models for Co-Design, 2003. MEMOCODE'03. Proceedings. First ACM and IEEE International Conference on*. 2003, pp. 123–132, IEEE.

[3] Shuvra S Bhattacharyya, Ed F Deprettere, Rainer Leupers, and Jarmo Takala, *Handbook of signal processing systems*, Springer Science & Business Media, 2013.

[4] Jerónimo Castrillon Mazo and Rainer Leupers, *Programming Heterogeneous MPSoCs*, Springer International Publishing, Cham, 2014.

[5] Texas Instruments, *Multicore DSP+ARM KeyStone II System-on-Chip (SoC) - SPRS866E*.

[6] Xilinx, *UltraScale Architecture and Product Overview (accessed 12/2015)*.

[7] Kalray, *MPPA MANYCORE: a multicore processors family*.

[8] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat, *Synchronization and linearity: an algebra for discrete event systems*, John Wiley & Sons Ltd, 1992.

[9] Neal Bambha, Vida Kianzad, Mukul Khandelia, and Shuvra S. Bhattacharyya, "Intermediate representations for design automation of multiprocessor DSP systems," *Design Automation for Embedded Systems*, vol. 7, no. 4, pp. 307–323, 2002.

[10] *PolyCore Software, Inc. - http://polycoresoftware.com/ (accessed 12/2015)*.

[11] *Silexica - https://silexica.com/ (accessed 12/2015)*.

[12] *Vector Fabrics - https://www.vectorfabrics.com/ (accessed 12/2015)*.

[13] Bart Kienhuis, Ed Deprettere, Kees Vissers, and Pieter van der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *Application-Specific Systems, Architectures and Processors,*

1997. Proceedings., IEEE International Conference on. 1997, pp. 338–349, IEEE.

[14] Johan Eker, Jörn W Janneck, Edward Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, Yuhong Xiong, et al., "Taming heterogeneity-the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[15] Maxime Pelcat, Slaheddine Aridhi, Jonathan Piat, and Jean-François Nezan, *Physical Layer Multi-Core Prototyping: A Dataflow-Based Approach for LTE eNodeB*, vol. 171, Springer, 2012.

[16] Hervé Yviquel, *From dataflow-based video coding tools to dedicated embedded multi-core platforms*, Ph.D. thesis, Rennes 1, 2013.

[17] Edward A Lee and David G Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, 1987.

[18] Edward A. Lee and Thomas M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, 1995.

[19] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya, "Functional DIF for rapid prototyping," in *Proceedings of the International Symposium on Rapid System Prototyping*, June 2008, pp. 17–23.

[20] W. Plishker, N. Sane, M. Kiemb, and S. S. Bhattacharyya, "Heterogeneous design in functional DIF," in *Proceedings of the International Workshop on Systems, Architectures, Modeling, and Simulation*, Samos, Greece, July 2008, pp. 157–166.

[21] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, February 1996.

[22] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*, Springer, second edition, 2013, ISBN: 978-1-4614-6858-5 (Print); 978-1-4614-6859-2 (Online).

[23] S. S. Bhattacharyya, R. Leupers, and P. Marwedel, "Software synthesis and code generation for DSP," *IEEE Transactions on Circuits and Systems — II: Analog and Digital Signal Processing*, vol. 47, no. 9, pp. 849–875, September 2000.

[24] S. Lin, L.-H. Wang, A. Vosoughi, J. R. Cavallaro, M. Juntti, J. Boutellier, O. Silvén, M. Valkama, and S. S. Bhattacharyya, "Parameterized sets of dataflow modes and their application to implementation of cognitive radio systems," *Journal of Signal Processing Systems*, vol. 80, no. 1, pp. 3–18, July 2015.

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

[26] George EP Box and Norman Richard Draper, *Empirical model-building and response surfaces*, vol. 424, Wiley New York, 1987.

[27] Fabrice Bellard, "QEMU, a Fast and Portable Dynamic Translator.," in *USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 41–46.

[28] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, and others, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[29] O. M. Group, "UML profile for MARTE: Modeling and analysis of real-time embedded systems, version 1.0,," .

[30] Stan Liao, Grant Martin, Stuart Swan, and Thorsten Grötker, *System Design with SystemC^{TM}*, Springer Science & Business Media, 2002.

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

[32] Masaki Gondo, Fumio Arakawa, and Masato Edahiro, "Establishing a standard interface between multi-manycore and software tools-SHIM," in *COOL Chips XVII, 2014 IEEE*. 2014, pp. 1–3, IEEE.

[33] Jianjiang Ceng, Weihua Sheng, Jeronimo Castrillon, Anastasia Stulova, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr, "A high-level virtual platform for early MPSoC software development," in *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*. 2009, pp. 11–20, ACM.

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

[35] Vida Kianzad and Shuvra S. Bhattacharyya, "CHARMED: A multi-objective co-synthesis framework for multi-mode embedded systems," in *Application-Specific Systems, Architectures and Processors, 2004. Proceedings. 15th IEEE International Conference on*. 2004, pp. 28–40, IEEE.

[36] Thierry Grandpierre, Yves Sorel, and Action OSTRE, "Un nouveau modèle générique d'architecture hétérogène pour la méthodologie AAA," *Actes des Journées Francophones sur l'Adéquation Algorithme Architecture, JFAAA'02*, 2002.

[37] Erwan Raffin, Christophe Wolinski, François Charot, Krzysztof Kuchcinski, Stéphane Guyetant, Stéphane Chevobbe, and Emmanuel Casseau, "Scheduling, binding and routing system for a run-time reconfigurable operator based multimedia architecture," in *Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on*. 2010, pp. 168–175, IEEE.

[38] Maxime Pelcat, Jean Francois Nezan, Jonathan Piat, Jerome Croizer, and Slaheddine Aridhi, "A system-level architecture model for rapid prototyping of heterogeneous multicore embedded systems," in *Proceedings of DASIP conference*, 2009.

[39] Adam Donlin, "Transaction level modeling: flows and use models," in *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. 2004, pp. 75–80, ACM.

[40] Thierry Grandpierre, *Modélisation d'architectures parallèles hétérogènes pour la génération automatique d'exécutifs distribués temps réel optimisés*, Ph.D. thesis, 2000.

[41] Endri Bezati, Richard Thavot, Ghislain Roquier, and Marco Mattavelli, "High-level dataflow design of signal processing systems for reconfigurable and multicore heterogeneous platforms," *Journal of real-time image processing*, vol. 9, no. 1, pp. 251–262, 2014.

[42] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J.-F. Nezan, and S. Aridhi, "Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming," in *Education and Research Conference (EDERC), 2014 6th European Embedded Design in*, Sept 2014, pp. 36–40.