A Model Driven Design Framework for High Performance Embedded Systems

Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Anne Etien, Rabie Ben Atitallah, Philippe Marquet, Jean-Luc Dekeyser

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Model Driven Design Framework for High Performance Embedded Systems

Abdoulaye Gamatié — Sebastien Le Beux — Éric Piel — Anne Etien — Rabie Ben-Atitallah —

Philippe Marquet — Jean-Luc Dekeyser

N° 6614

August 2008

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6614--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche LILLE - NORD EUROPE

A Model Driven Design Framework for High Performance Embedded Systems

Abdoulaye Gamatié* , Sebastien Le Beux† , Éric Piel‡ ,
Anne Etien§ , Rabie Ben-Atitallah¶ ,
Philippe Marquet‖ , Jean-Luc Dekeyser**

Thème COM — Systèmes communicants
Équipes-Projets DaRT

Rapport de recherche n° 6614 — August 2008 — 44 pages

Abstract: Modern embedded systems integrate more and more complex functionalities. At the same time, the semiconductor technology advances enable to increase the amount of hardware resources on a chip for the execution. High performance embedded systems specifically deal with the optimized usage of such hardware resources to efficiently execute their functionalities. The design of high performance embedded systems mainly relies on the following challenging issues: first, how to deal with the parallelism in order to increase the performances; second, how to abstract their implementation details in order to manage their complexity; third, how to refine these abstract representations in order to produce efficient implementations.

This paper presents the GASPARD design framework for high performance embedded systems as a solution to the above issues. GASPARD uses the repetitive Model of Computation (MoC), which offers a powerful expression of the parallelism available in both system functionality and architecture. Embedded systems are designed at a high abstraction level with the MARTE (Modeling and Analysis of Real-time and Embedded systems) standard profile, in which our repetitive MoC is described by the so-called Repetitive Structure Modeling (RSM) package. Based on the Model-Driven Engineering (MDE) paradigm, MARTE models are refined towards lower abstraction levels, which make possible the design space exploration. By combining all these capabilities, GASPARD

* *Abdoulaye.Gamatie@lifl.fr*
† *Sebastien.Le-Beux@lifl.fr*
‡ Technical University of Delft, The Netherlands. Email: *e.a.b.piel@tudelft.nl*
§ *Anne.Etien@lifl.fr*
¶ *Rabie.Ben-Atitallah@lifl.fr*
‖ *Philippe.Marquet@lifl.fr*
** *Jean-Luc.Dekeyser@lifl.fr*

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne
40, avenue Halley, 59650 Villeneuve d'Ascq
Téléphone : +33 3 59 57 78 00 — Télécopie : +33 3 59 57 78 50

allows the designers to automatically generate code for formal verification, simulation and hardware synthesis from high level specifications of high performance embedded systems. Its effectiveness is demonstrated with the design of an embedded system for a multimedia application.

Key-words: Embedded system design, High performances System, Model Driven Engineering, MARTE standard profile, parallel programming, Repetitive Structure Modeling.

Un environnement basé sur l'ingénierie dirigée par les modèles pour la conception de systèmes embarqués hautes performances

Résumé : Les systèmes embarqués modernes intègrent de plus en plus de fonctionnalités complexes. Dans le même temps, les avancées technologiques dans les semi-conducteurs permettent d'augmenter la quantité de ressources matérielles sur une puce. Les systèmes embarqués à hautes performances doivent optimiser l'utilisation de telles ressources en vue d'une exécution efficace de leurs fonctionnalités. La conception de ces systèmes pose principalement les questions suivantes : comment traiter le parallélisme afin d'augmenter les performances ; comment abstraire leurs détails d'implémentation pour venir à bout de leur complexité ; comment raffiner ces représentations abstraites pour produire à terme des implémentations efficaces.

Ce rapport présente l'environnement de conception GASPARD pour les systèmes embarqués à hautes performances comme une solution aux questions soulevées ci-dessus. GASPARD utilise un modèle de calcul répétitif, qui offre une forte expressivité du parallélisme présent à la fois dans les fonctionnalités et l'architecture d'un système. Les systèmes embarqués sont conçus à un niveau d'abstraction élevé à l'aide du profil standard MARTE (*Modeling and Analysis of Real-time and Embedded systems*), dans lequel notre modèle de calcul répétitif est décrit par le paquetage *Repetitive Structure Modeling (RSM)*. En se basant sur l'Ingénierie Dirigée par les Modèles (IDM), les modèles MARTE sont raffinés vers des niveaux d'abstraction plus bas, rendant possible l'exploration de l'espace de conception. En exploitant tout ce potentiel, GASPARD permet aux concepteurs de générer automatiquement du code pour la vérification formelle, la simulation et la synthèse de matériel à partir de spécifications haut niveau de systèmes embarqués à hautes performances. Son efficacité est démontrée ici au travers de la conception d'un système embarqué dédiée à une application multimédia.

Mots-clés : Conception de systèmes embarqués, système hautes performances, ingénierie dirigée par les modèles, profil standard MARTE, programmation parallèle, modélisation de structures répétitives.

1 Introduction

Modern embedded systems are becoming more and more sophisticated and resource demanding. The concerned application domains are multiple: state-of-the-art multimedia applications such as video encoding/decoding, software-defined radio, detection systems such as radars, sonars, or telecommunication systems such as mobile phones, antennas, etc. The need to meet *performance* requirements in these systems leads to the recent advances in technology, which enable to integrate on a single chip an increasing number of transistors (up to a billion today). In this aim, *System-on-Chip* (SoCs) have been very promising. Such a system can manage at the same time several applications or functionalities. It consists of software and hardware parts. The hardware part is composed of components such as processors, memory, scalable interconnection network and hardware accelerators. The software part defines the functionality of the system. Typically, for multimedia applications, the functionality consists in systematically applying specific filters (*e.g.* discrete cosine transform, fast fourier transform) to video/image streams.

Because of the physical restrictions in terms of frequency and voltage, which have a real impact on the computational performances, the expansion of the processing power of a SoC requires to put *multiple* and possibly heterogeneous processors or cores into a single chip. For this reason, the exploitation of the parallelism available in *Multi-Processor System-on-Chip* (MPSoCs) architectures is a very attractive solution for the execution of high performance applications.

1.1 Design Challenges

This section discusses some critical design challenges faced by designers of high performance embedded systems. Some of these challenges concern embedded systems in general and are largely discussed in literature [44]. From these discussions, a number of important requirements are identified as a basis to define well-suited design methodologies for these systems.

Need to Deal with Parallelism In the current industrial practice, hand-coding is still widely adopted in the development of embedded systems. Parallelism is managed at a low level by different expert teams, which have a deep knowledge of the system (both hardware and software parts) in order to meet the performance requirements. Hand-coding is clearly not suitable for an efficient development of large embedded systems because it is very tedious, error-prone and expensive. Another way to deal with parallelism consists in using general parallel programming models such as the distributed memory programming model *Message Passing Interface* (MPI) [33] or the shared memory model OpenMP [1]. Both programming models provide developers with a set of directives and library of routines that are used to express parallel computations in general. They assume an abstraction layer between the applicative level and the hardware architecture level, which serves as an execution support for the parallelization. As a result, the performances of the overall system strongly depends on this intermediate layer. MPI is a rich model that enables to program the distribution of both data and computations respectively onto available memory and processors. However, to obtain efficient programs regarding high performance, very careful manual optimizations are required. OpenMP enables

to program only the distribution of computations on processors. Data distribution is not under the control of the programmer. As a consequence, typical constraints of embedded systems, such as memory usage are hardly satisfied.

From the above observations, one can understand that the programming of large and complex parallel embedded systems with MPI and OpenMP is also a very difficult task. Moreover, in the case of SoCs, the architecture itself has to be specified in order to satisfy particular design constraints. OpenMP and MPI offer very general programming mechanisms that do not enable to suitably answer this specific need. For all these reasons, we believe that the design of high performance embedded systems particularly calls for new expressive *parallel programming paradigms*. Such paradigms should provide designers with some efficient way to *separately* represent, at a high level, all the potential parallelism that is inherent to both software applications and hardware architectures. Furthermore, they should be rich enough to explicitly express different mapping possibilities of the application on the architecture, taking into account the parallelism in embedded systems. Finally, in order to optimize the execution performances, the intermediate layers between the applicative level and the hardware architecture should be removed as much as possible.

Need of Abstract Models The design of SoC is facing today a strong pressure on reducing *time-to-market* while the *complexity* of these systems has been increasing. In addition to this dilemma, we notice that the initial cost for the physical realization of the SoC (the mask creation of the chip) is very expensive. Such an outlay strongly imposes a more careful development of prototypes for design analysis and validation. System developers have to rely on some *costless means* allowing them to simulate and analyse the behavior of designed systems before their realization.

Design abstraction offers a possible solution to address the above issues concerning the time-to-market and complexity dilemma, and the SoC development cost. More concretely, one needs *models that capture the strict relevant information depending on the required abstraction level*. The global complexity of a system is addressed from multiple viewpoints or abstraction levels, so that one is able to easily focus on some specific aspects. Abstract models favor an efficient design reuse, typically through incremental *refinements* from higher level models to lower level models. Here, by refinement, we mean a transformation that makes a given model more concrete w.r.t. a target representation (which is in general more precise than its current representation). On the other hand, since models are often executable and verifiable, they also serve as an interesting support for both behavioral simulation and property analysis without having necessarily the actual implementation of systems. In some cases, they are even used to automatically synthesize this implementation. Finally, abstract models enable to deal with the heterogeneity of a system since its components can be manipulated at high description levels that suitably abstract away the specific details of each component.

Need of Seamless Methodologies The development of a SoC usually starts with the concurrent design, or *co-design*, of both software application and hardware architecture. Then, the application part is mapped onto the hardware part, during the association phase. Finally, simulation models from various ab-

straction levels are generated for the whole system. The different aspects of this development process are potentially handled by *different domain experts who must communicate safely* in order to achieve the resulting design. In such a context, the design and analysis activities become very difficult due to the ever increasing complexity of SoCs. As a consequence, the *productivity* of designers strongly gets penalized. Another critical aspect concerns the design space exploration; *i.e.* how the analysis and the simulation results obtained from the different abstraction levels are exploited for an efficient redesign by modifying the high level system models. So, an important challenge here is to find *design methodologies* with supporting tools that adequately address all these issues concerning large and complex embedded systems.

1.2 Our Proposition: the GASPARD Framework

In this paper, we present our design *framework*, GASPARD (*Graphical Array Specification for Parallel and Distributed Computing*), as a solution to the development of high performance embedded systems addressing the above challenges. Here, by framework, we mean an environment that provides designers with at least the following means: a formalism for the description of embedded systems at a high abstraction level, a methodology covering all system design steps, and a tool-set that supports the entire design activity.

The design of SoCs in GASPARD specifically relies on the *repetitive* model of computation (MoC) [12], which offers a very suitable way to express and manage the potential parallelism in a system. This MoC is inspired by Array-OL [21], a domain-specific language originally dedicated to intensive signal processing applications. It extends the basic notions of this language and offers an elegant and very expressive way to describe both *task parallelism* and *data parallelism*, and the combination of both.

The repetitive MoC is used in GASPARD, via the MARTE standard profile and more precisely its RSM package, to describe the parallel computations in the application software part, the parallel structure of its hardware architecture part, and the association of both parts. The resulting abstract models are afterwards deployed towards specific implementations. Finally, different automatic refinements from the higher abstraction level are defined, according to Model Driven Engineering (MDE) paradigm, towards lower levels for various purposes: simulation at different abstraction levels with SystemC [4], hardware synthesis with VHDL [29], formal validation with synchronous languages [23], high performance computing with OpenMP Fortran and C [48]. MDE enables to clearly identify different intermediate abstraction levels. Thus, it facilitates the decomposition of the refinement process into successive steps. In this paper, we mainly consider the first three facilities to deal with the design space exploration of high performance embedded systems.

1.3 Outline of the Paper

The remainder of this paper is organized as follows: Section 2 first presents some related works. Section 3 describes the general design of a multimedia embedded system considered along the paper to illustrate our design framework. Section 4 introduces the repetitive MoC and the MDE paradigm which are the

foundations of the GASPARD modeling concepts presented in Section 5. Section 6 addresses the refinement of high level GASPARD models: it describes the different transformation chains implemented towards various analysis and simulation target technologies. A multi-level design space exploration methodology is proposed for GASPARD in Section 7. This methodology is illustrated by some implementation results on our running example in Section 8. Finally, concluding remarks are given in Section 9.

2 Related Works

We first present some *high performance programming models* that deal with parallelism. Then, we discuss methodological aspects through two outstanding paradigms: *Platform-Based Design* (PBD) and *Model-Driven Engineering*. In both paradigms, abstract models and model refinements are central.

2.1 High Performance Programming Models

The programming of high performance systems has been extensively investigated through the last decades. Among the proposed solutions, we have already mentioned the parallel programming models offered in MPI [33] and OpenMP [1].

More recent languages are StreamIt [49] and the DARPA high productivity languages Chapel [15], Fortress [3] and X10 [17]. Their main objective is to improve productivity in high performance scientific computing by providing programmers with suitable macro constructs. The efficient implementation of these constructs on parallel architectures can be however expensive. In these languages, compilers play a central role regarding the production of efficient code. In particular, their optimisation is a real issue to minimize the decrease of performances that results from the interpretation of the macro constructs [30]. Furthermore, although these languages provide higher abstractions compared to usual programming languages, they consider specific architecture models. Thus, similarly to MPI and OpenMP, they also have the inconvenient to be not well adapted for SoC design in which one needs to program specialized architectures.

An interesting high level language is the data-parallel formalism ALPHA [52], which is very close to Array-OL. It manipulates polyhedra instead of arrays. This leads to different specification styles. ALPHA particularly suits for the specification of *systolic* architectures. As a result, it does not offer a satisfactory solution to the design of other types of architecture models as it is needed here for SoCs.

2.2 Platform-Based Design for Embedded Systems

There are several co-design methodologies for embedded systems that start from high level designs from which the system implementation is produced after some automatic or manual refinements. The PBD methodology [45] is a typical example. Its main idea is to facilitate the design task by enabling successive refinements of high level specifications of system functionality and architecture with reusable components so as to rapidly meet the implementation requirements of the system. The principles of PBD is found in frameworks such as Metropolis [6] of Berkeley, VCC [32] of Cadence, the Artemis workbench [41],

and CoFluent studio [16]. Except the former, which considers several MoCs, each of these frameworks adopts a particular MoC and provides the designers with a library of domain-specific components for platform instantiation. While PBD helps to shorten the time-to-market and to reuse verification, it may potentially reduce the flexibility of design since the space of choices is limited to the available system components. Furthermore, in the case of SoCs, which are composed of heterogeneous components that are often developed with dedicated tools (*e.g.* an ARM processor with the ARM environment, a hardware accelerator with a high level synthesis tool), it is highly desirable to have a *single design model* that covers the whole system description. But, most of PBD approaches combine different design models that capture various aspects of a system. For instance, the Artemis workbench that is devoted to multimedia domain considers Simulink representations for functionality description and Kahn Process Networks (KPN) MoC for architecture description.

2.3 Model Driven Engineering for Embedded Systems

Another interesting design paradigm that is very close to PBD is the already mentioned MDE approach. It has been increasingly adopted for the design of embedded systems in general [46]. The basic modeling formalism is the general purpose language UML (Unified Modeling Language), which offers attractive graphical specification concepts. Because of its generality, UML is refined by the notion of *profile* to address domain-specific problems.

There are currently several profiles for the design of embedded systems such as SysML [38], UML SPT [37], UML-RT [47], TUT Profile [26], ACCOR/UML [27] and Embedded UML [31]. Among these profiles, only the former two have been standardized by the Object Management Group (OMG). SysML is a general-purpose modeling language for system design, while UML SPT is dedicated to the modeling of time, schedulability, and performance-related aspects of real-time systems. The UML-RT and ACCOR/UML profiles are also dedicated to real-time systems. However, they are less rich in terms of concepts than UML SPT. The Embedded UML profile has been defined within VCC as an experimental proposal that goes beyond the real-time field. It also includes aspects from the hardware/software co-design field. This last field is mainly taken into account in the TUT Profile, which defines concepts allowing one to model applications, platforms and their mapping. Among the few profiles that specifically focus on SoC modeling, we mention UML4SystemC [42] and the OMG UML4SoC profile [24]. They offer an abstraction of the Register Transfer Level (RTL). This abstraction accelerates the simulation of the software and hardware parts of a system. Because all these profiles may potentially overlap, significant standardization efforts have been recently realized by the OMG, resulting in the single unified and effective MARTE standard profile [36], on which GASPARD relies. MARTE stands for Model and Analysis Real-Time Embedded system. It is an evolution of the UML SPT profile and borrows some concepts from the more general SysML profile.

While these profiles allow to specify a system with high level models, refinements from such models towards low level models have to be achieved. Typically, from the specification of an embedded system with a profile, one would like to generate executable implementations of the system. For instance, the UML4SoC and UML4SystemC profiles allow one to automatically generate SystemC or

SystemVerilog code. Since the concepts manipulated in these profiles are very close to the implementation level, they are directly mappable to the targeted language ones. This reduces the flexibility when considering other targets because the overall mapping has to be redefined. Some alternative propositions use specific notations instead of using profiles, defining an entirely *executable* model semantics [2, 34, 43]. Such expressive notations allow one to define models with sufficient information so that the specified system can be completely generated. However, here also, the code is directly generated from the specifications, without any intermediary representation. The same is observed in the VHDL code generation from UML [11, 19], where the code is obtained directly by mapping the UML concepts with the VHDL syntax. More generally, this absence of successive refinements leads to a lack of flexibility when targeting new abstraction levels or new languages. While these approaches rely on an abstraction of the system by using high level models, they only exploit a little of its benefits by directly being dependent on target languages or abstraction levels.

3 A Typical Embedded System

A typical embedded system dedicated to signal processing application is presented in this section. Our objective is to illustrate the design challenges faced by developers in terms of architecture topology, application task mapping, hardware and software component reuse. We consider the H.263 video codec standard [18]. It is suitable for low bit rate wireless video systems. In particular, it is useful for recent applications in cellular videophones, wireless surveillance systems, or mobile patrols. We focus on the encoder part which performs the most intensive computations. Most of the examples used in this paper are extracted from this application.

This H.263 encoder application sketched in Fig. 1 is composed of three tasks: the Discrete Cosine Transform (DCT) computation, the quantization and the coding. The DCT task transforms the data, here frame pixels, into spatial frequency coefficients. The quantization (QUANT) task approximates the DCT coefficients by a small set of possible values. Finally, the Huffman Coding (HC) performs a data compression by assigning a short binary word to the most frequent values.

The application takes as input a stream of QCIF frames (176×144 pixels in the YCbCr format). In the encoding algorithm, data are handled by *macroblocks*, as illustrated in Fig. 2. A macroblock corresponds to a 16×16 pixel area of a video frame. It is represented in the YCbCr format, which contains a luminance component (Y), a blue chrominance component (Cb), and a red chrominance component (Cr). Luminance blocks describe the intensity, or brightness, of pixels, whereas chrominance blocks define the color of pixels. A macroblock contains six 8×8 blocks: four blocks contain luminance values, one block contains blue chrominance values, and one block contains red chrominance values. Since the processing of each macroblock can be done independently from the others, this data structure confers to the H.263 algorithm a potential parallelism in terms of data computation.

The execution of such an application in an embedded system implies several constraints. In the specific case of high performance embedded systems,

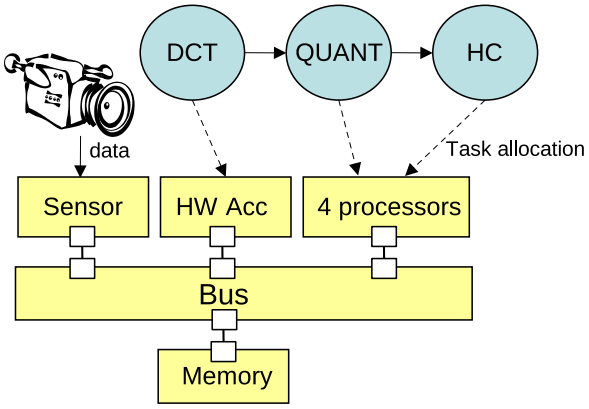

Figure 1: Informal representation of the H.263 encoder application.

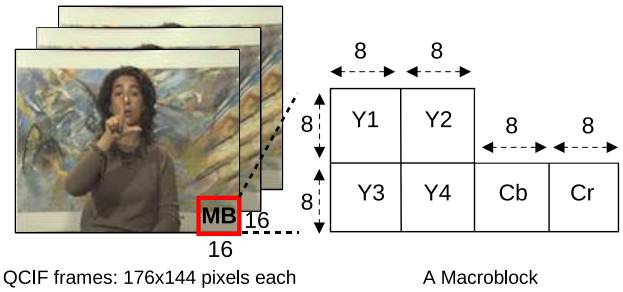

Figure 2: Structural decomposition of data in the frames.

the constraints related to the execution time are highly crucial. For instance, depending on the frame rate to achieve (10 or 30 frames per second), the same H.263 encoder application is performed more or less efficiently. Hence, the architecture executing the application has to be designed carefully, so as to meet the application's performance requirements.

A well-known way to increase performances is to execute as many as possible parts of the application in parallel. So, the architecture executing the H.263 encoder has to exploit the potential parallelism of the application. Fig 1 represents a possible architecture including four processors and a hardware accelerator as computing resources, a memory as storage resource, and a bus for the communications between these resources. The DCT task is mapped onto the hardware accelerator and the workload of quantization and coding tasks is equitably shared between the four processors. Indeed, the DCT time consuming task is efficiently performed using hardware accelerators because of the regularity of its corresponding algorithm. Moreover, the mapping of the two other tasks on processors is relevant since it adequately exploits the potential parallelism provided by the data decomposition into macroblocks (each macroblock is managed by a given processor).

The architecture and the mapping sketched in this figure are only one possible implementation solution. For instance, an architecture only composed of processors as computing resources could be an interesting alternative. The

higher number of processors would compensate the lack of computing power previously provided by the hardware accelerator.

The choice among these various solutions has a strong impact on the performances of the resulting embedded system. Nowadays, such a choice mostly relies on the knowledge of embedded system designers. The specifications of the application and the architecture are often thought and implemented together at a low abstraction level. The first raw implementation thus often constrains the following ones, drastically reducing the explored design space. This may hopefully lead to an acceptable solution but, in the worst cases, it leads to a solution that does not achieve the performance requirements. In this case, the whole embedded system has to be completely rethought. Such a methodology based on designer's know-how leads to an overhead of the design time and increases the cost of the embedded system. These drawbacks are increased by the ever growing complexity and even higher performance requirements of applications. In addition, from the architecture point of view, the technologies allow one to embed more resources (*e.g.* computing, storage) on a single chip. Therefore, the design space accordingly becomes larger. The current design methodologies are no more adapted to address the exploration of this wide design space, yet essential to design high performance embedded systems.

To solve this bottleneck, we believe that it is useful to manage the three challenges mentioned in Section 1. It is necessary to exploit the potential parallelism available in applications when designing the architecture and the mapping of the former on the latter. High abstraction level representations of embedded systems allow one to deal with their complexity. A methodology in which implementations are generated from such representations enables to rapidly explore wide design spaces without the above drawbacks of manual implementations. These aspects are key-points for the design of tomorrow's high performance embedded systems.

4 Foundations

This section introduces the basic concepts on which GASPARD relies. We first introduce the repetitive MoC, which allows the expression of the potential parallelism available in high performance embedded systems. Such systems are specified in a factorized way and at a high abstraction level. Then we present the MDE paradigm that offers the mechanisms used to refine high level representations and to generate their corresponding implementations. Finally, we briefly present the Repetitive Structure Modeling part of the MARTE standard that allows one to represent the concepts of the repetitive MoC, and to exploit it with an MDE approach.

4.1 The Repetitive Model of Computation

Modeling high performance embedded systems requires concepts that allow us to describe the regularity of a system structure in a factorized way. Such concepts provide the designer with a way to efficiently and explicitly express models with a high number of identical components. The repetitive structures are the basis of the repetitive MoC originally inspired by Array-OL, which is dedicated to intensive multidimensional signal processing [21]. They have been extended,

in the repetitive MoC, to describe non-applicative regularity, *i.e.* hardware architectural topologies. The following description of the repetitive MoC is independent from application and architecture considerations.

A structural element T is replicated into several structural elements. The *repetition space* is defined by a vector. The number of iterations is determined by multiplying the coordinates of this vector. The left-hand side of Fig. 3 represents a repetitive structure. H is a hierarchical structure, the dashed box t:T is a repeated structural element. {2,2} corresponds to the repetition space of this structural element. The right-hand side corresponds to the unrolled equivalent representation. Each dashed box $t_{i,j}$:T corresponds to a structural element associated with a given iteration in the repetition space.

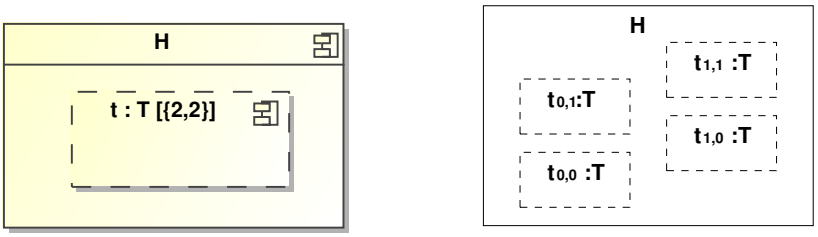

Figure 3: Factorized (left) and unrolled (right) regular structure.

Structural elements handle information in the form of arrays materialized by ports. The shape of a port defines the shape of the corresponding array and specifies how the information are structured. Dependencies are defined between the arrays handled by different structural elements. A repeated structural element manipulates subsets of such arrays, which are referred to as *patterns*. The dependencies between arrays and patterns are expressed with the three repetitive link topologies, Tiler, Reshape and InterRepetition, detailed below.

4.1.1 Tiler

A Tiler connector expresses how a multidimensional array is tiled by patterns. For this purpose, it connects an array to the patterns of a repeated structural element, as illustrated on the top of Fig. 4. In this example, i1 and o1 are the ports of the H structural element. They represent the multidimensional arrays of H. The port i2 and o2 represent the patterns of the t:T repeated structural element.

Boulet [12] gives a formal description of tilers. In this paper, we adopt an intuitive approach to explain how tilers work. The bottom part of Fig. 4 is an equivalent but not factorized expression of the Tilers illustrated on the top. All the repetitions (*e.g.* $t_{0,0}$:T, $t_{0,1}$:T) of the repeated structural element are made explicit. Each one manipulates its own patterns: the pattern corresponding to i2 is a bi-dimensional array whereas the one corresponding to o2 is a scalar. The patterns are build according to the arrays of the H structural element, as indicated via arrows. The 2×2-patterns i2 are constructed from elements contained in the 3×3-array i1. Symmetrically, the o2 scalars are used to build the 2×2-array o1.

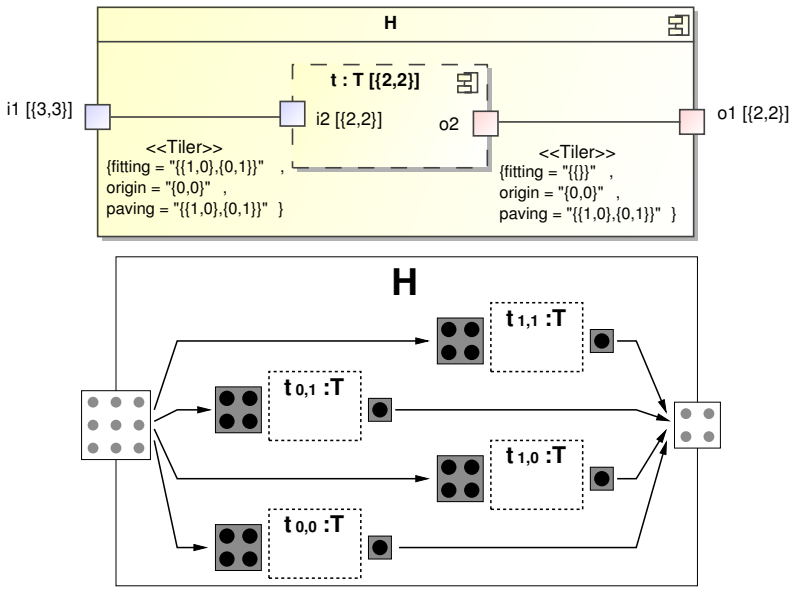

Figure 4: Tilers express multiple link topologies in a factorized way.

The way patterns are built from an array depends on a *tiling operation* which requires the origin, paving and fitting attributes. The origin vector specifies the origin of the reference pattern in the array. The paving and fitting matrices respectively specify how an array is covered by patterns and how the patterns are constructed with array elements. For more details, see [12]

Fig. 5 precisely describes the dependencies between patterns and array elements. The vector r denotes iteration steps in the repetition space: each $r = \binom{i}{j}$ in Fig. 5 identifies the corresponding $t_{i,j}$:T in Fig. 4. For each iteration, the elements highlighted in the 3×3-array i1 are used to build the corresponding 2×2-pattern i2. For example, the four elements on the bottom left-hand side of the array are used to build the pattern of the $t_{0,0}$:T structural element. Thus, the scalar element in the middle of the array is used to build each of the four patterns. In the same way, the right-hand side of Fig. 5 illustrates the dependencies between o2 and o1.

According to these attributes, a Tiler expresses dependencies between a M-dimensional array and N-dimensional patterns. These dependencies are not limited to compact and parallel to axis patterns (*e.g.* rectangular shaped patterns). Dependencies are usually expressed via indexes in languages. It is the case in ALPHA [52]. The explicit manipulation of indexes in specifications is tedious and error prone: the implied complexity dramatically increases with the number of dimensions and the shape of patterns. Tilers avoid these drawbacks.

4.1.2 Reshape

A Reshape enables to express complex link topologies in which the elements of a multidimensional array are redistributed in another multidimensional array. For this purpose, a Reshape connector links two ports of structural elements included

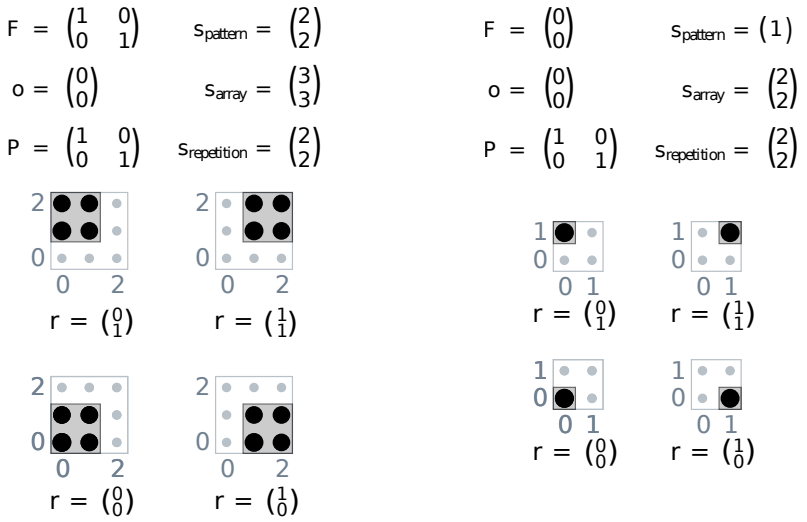

Figure 5: Dependencies between patterns and array elements.

in the same hierarchical structural element (as opposed to the Tiler which links a port of a repeated element to those of a hierarchical one). In fact, a Reshape is a combination of two Tilers. It enables the factorisation of complex dependencies between two arrays. The top of Fig. 6 represents a Reshape connector that expresses dependencies between the o and i ports.

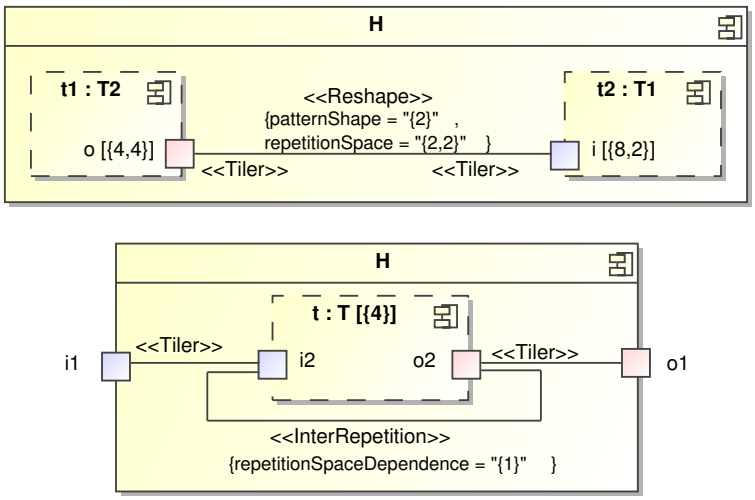

Figure 6: Representation of the Reshape (top) and the InterRepetition (bottom) dependencies.

4.1.3 InterRepetition

An InterRepetition specifies dependencies between the repetitions of a given repeated structural element. This connector links a pattern of a repeated structural element with another pattern of the same repeated structural element, as illustrated in the bottom part of Fig. 6. The usefulness of the InterRepetition link topology is illustrated later in the paper.

Some of the concepts presented in this section originally come from Array-OL. They have been extended to express regularity and parallelism in the whole embedded systems. Following the same objective, other concepts have been introduced in order to factorize complex dependencies. The next sections present a way to represent such concepts and then to refine them towards executable code.

4.2 The Model Driven Engineering Paradigm

MDE relies on the *model* concept. A model is an abstraction of a system in which all non-relevant details are taken away. It can be considered as a partial view of a system with a particular prism. It is always constructed with a specific purpose in mind and has not the vocation to represent the system in its whole. The semantics of concepts and relations handled in a model has to be precisely specified. This is the role of the *metamodel*, which states what valid models express: a model is conform to a metamodel.

The most known metamodel is certainly UML, firstly normalised in 1997 by the OMG (Object Management Group). UML has become the common modeling language of the software development community. It gathers lots of useful concepts and their associated relations for software design. UML has the advantage to be widely spread and several modeling tools supporting a graphical representation of UML models have been developed. However, UML is too generic to represent specific aspects such as real-time constraints in embedded systems. Furthermore, some important concepts are missing. To overcome this limitation, the UML specification permits to create extensions, called *profile*. A profile is a set of stereotypes that specialize UML concepts. It has the advantage of allowing the usage of UML tools. Thus, a model respecting a specific profile is often the entry point for the users.

MDE has also the particularity to make models "executable". Indeed, models are no more "contemplative" as they were formerly, but they are successively transformed until reaching code. A model *transformation* is composed of *rules*. Each rule defines the way a set of concepts of a source metamodel is transformed into a set of concepts of the target metamodel. The target metamodel is usually more specific than the source one and manipulates concepts closer to the code. Successive transformations in which the target metamodel of one transformation is the source of the following one is a *transformation chain*. Transformation chains often lead to code, the last transformation is thus a model to text transformation corresponding to the code generation.

4.3 The MARTE RSM Package

Following the MDE paradigm, the concepts manipulated in our repetitive MoC are defined in a metamodel or a profile. This is the aim of the MARTE profile,

which has been recently standardized for the modeling of real-time embedded systems. Subsets of the profile enable the modeling of software application and hardware architecture. A subset, called RSM package, is based on our repetitive MoC and is dedicated to the modeling of regular structures (either in application or architecture). All the concepts we focus on are clearly identified in this standard. Fig. 7 depicts the basic stereotypes associated with the RSM package. The Tiler stereotype, situated on the bottom right-hand side, owns the origin, paving and fitting attributes necessary to realize tiling operations. Furthermore, the RSM package indicates that the Reshape stereotype is composed of two Tilers identified with the srcTiler and the targetTiler relations. A complete description of all these concepts is provided in the MARTE specification [36].

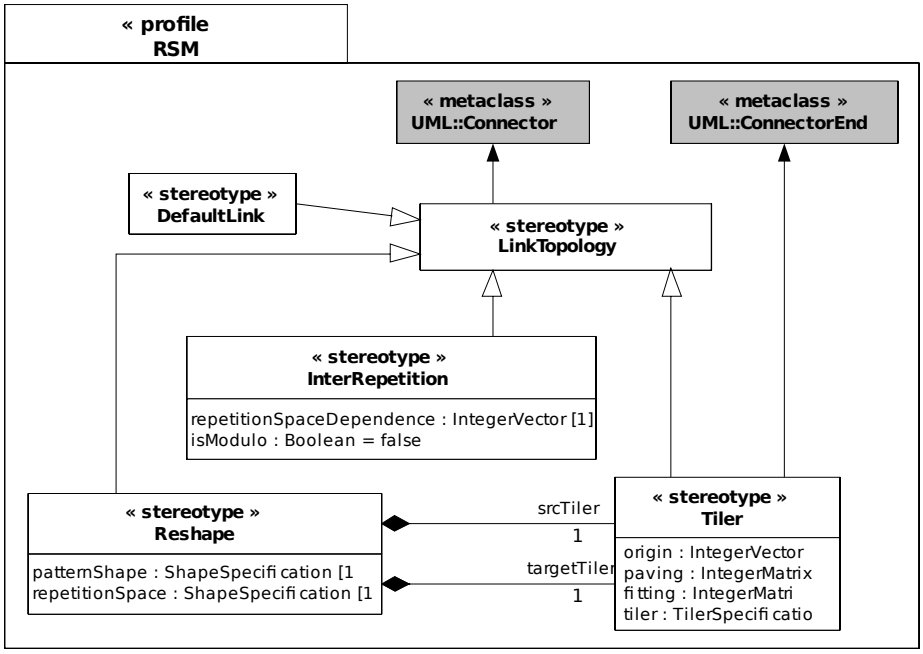

Figure 7: The RSM package of MARTE.

The overall MARTE profile is built in the same manner. Each concept necessary for the description of an embedded system is clearly identified with a stereotype, a relation or an attribute. This offers the possibility to model complex embedded systems at a high abstraction level. In the following section, we demonstrate the suitability of MARTE for the modeling of high performance embedded systems.

5 Modeling Concepts

This section presents the concepts provided by MARTE for the description of intensive signal processing applications, hardware architecture and the mapping of the former on the latter. Additional information corresponding to implementation details are provided through a deployment mechanism. All these concepts

are illustrated in the modeling of an embedded system dedicated to the execution of the H.263 encoder.

5.1 Application Functionality

5.1.1 Components

The targeted application functionality generally consists of data intensive computation algorithms as in the H.263 encoder application. There are basically three kinds of task components that are used to specify application functionality. An *elementary* component defines a task as an atomic function. A *repetitive* component expresses a data-parallel task in an algorithm. Finally, a *hierarchical* component enables to specify complex functionalities in a modular way; in particular, task parallelism is described using such a component.

We introduce below the H.263 encoder application designed with the MARTE profile. The interested readers can find a complete description of this application model in [40].

5.1.2 Modeling the H.263 Encoder

In Fig. 8, the H263Encoder component corresponds to the task that reads a QCIF frame in order to produce a compressed frame. A QCIF frame is decomposed into the three arrays materialized by the ports lumin, cbin and crin (one port for the luminance and two ports for the chrominance). The shape of a port defines the shape of the corresponding data array. For instance, the shape of the port lumin is 176×144. The compressed frame produced is decomposed into arrays materialized by the ports mbout and size, denoting respectively the compressed data and the compression level.

In the application part of a MARTE model, a repetition space on a task expresses data parallelism. On Fig. 8, the multiplicity {11,9} associated with the H263mb instance of the H263MacroBlock component (noted as H263mb:H263MacroBlock) denotes such a data parallel task. Each repetition of the task H263mb consumes three input patterns (lumin, crin and cbin) and produces two output patterns (mbout and size). A pattern corresponds to a subset of the overall frame. Each iteration step within the task repetition space consumes and produces patterns. Their construction relies on the data dependencies expressed via the Tiler connectors.

Fig. 9 represents a partial view of the data dependencies between the luminance part of a frame and the luminance part of a macroblock. They are expressed with the Tiler connecting the lumin port of the H263Encoder component to the lumin port of the H263mb component instance. The 16×16 pattern corresponds to the luminance component of a macroblock. The 176×144 array corresponds to a QCIF frame tiled 99 times (*i.e.* obtained by calculating the product of the vector dimensions corresponding to the {11,9} repetition space) by the pattern in order to produce all macroblock luminance components. The blue and red chrominance components of the macroblocks are extracted in the same manner.

The hierarchical component H263MacroBlock is formed of three tasks, which correspond to the three steps of the H.263 encoder algorithm: DCT, QUANT and HC. The DCT component contains both data and task parallelisms. The

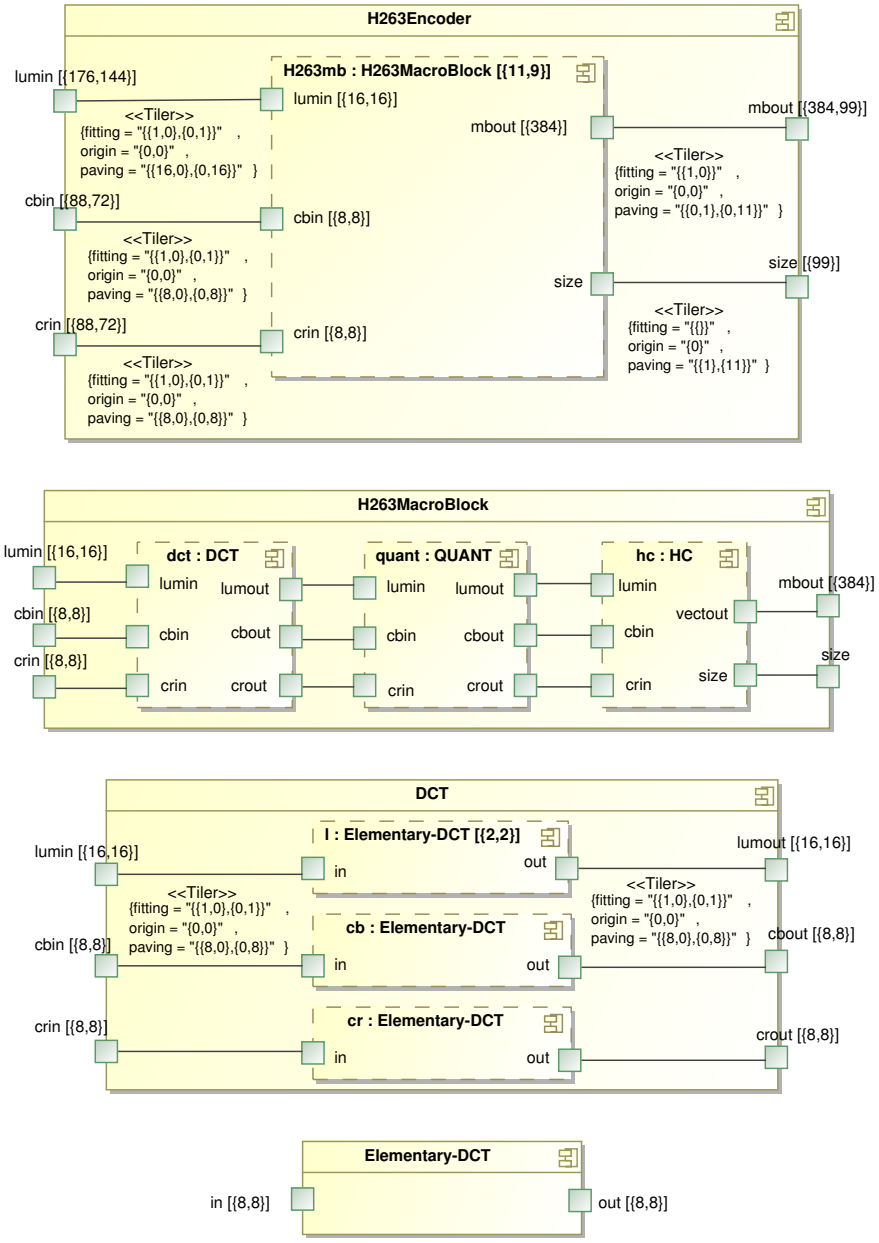

Figure 8: The main application components of the H.263 encoder.

aim of this component is to apply the DCT transformation algorithm to the luminance and chrominance macroblocks. For this purpose, the Elementary-DCT task is instantiated. It consumes a 8×8 array of pixels in order to produce a 8×8 array of spatial frequency coefficients. Since this task is elementary, its behavior is provided according to additional information included in the deployment described later on.

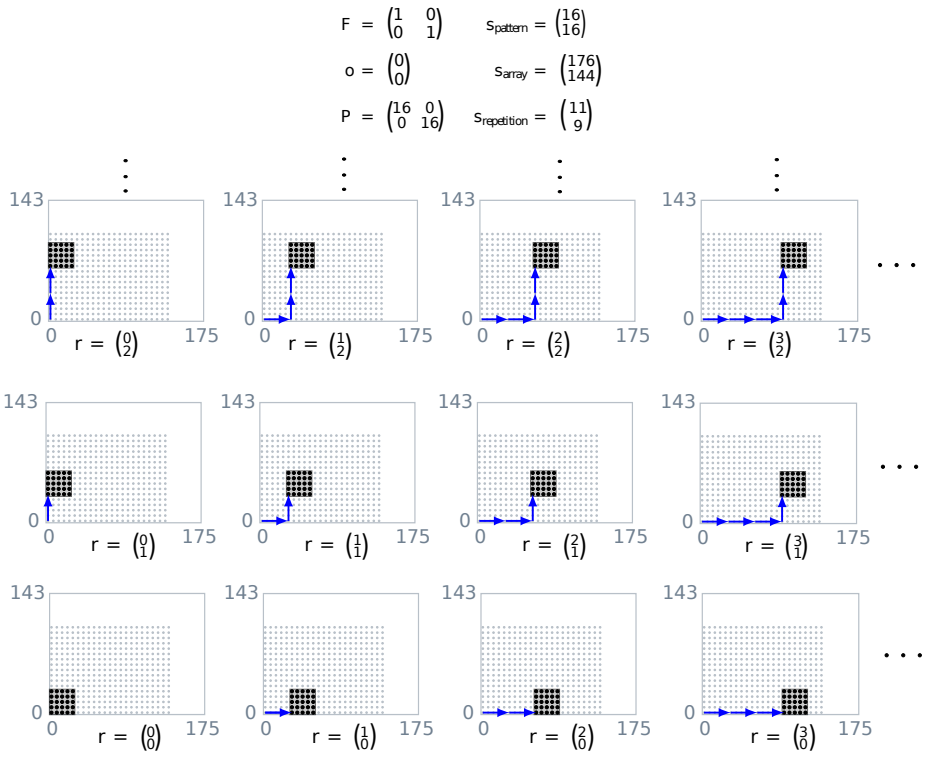

Figure 9: Tiling the luminance pixels of a QCIF frame.

At the top level of the application model, the received video streams are encoded according to a global repetition around the H263Encoder component. This repetition infinitely applies the encoding algorithm to the successive video frames.

This example demonstrates the effectiveness of MARTE for representing intensive signal processing applications. Indeed, the data parallelism, the task parallelism and the combination of both is efficiently expressed in the model.

5.2 Hardware Architecture

5.2.1 Architecture Modeling

Compared to UML, the MARTE profile allows one to describe hardware characteristics in a more precise way. The hardware architecture is described in a structural way. The notion of component permits to reuse a set of basic components in different places. In GASPARD, only the HW_Logical sub-package of MARTE is used. It describes typical components (*e.g.* HwRAM, HwProcessor, HwASIC, HwBus), their non functional properties (*e.g.* operating frequency, power consumption).

Similarly to application, the modeling of the hardware architecture is also described using the repetitive concepts of the MARTE RSM package. This is very useful when describing a grid of processing elements, such as the Tile64 architecture of Tilera [51]: the model is kept factorized.

5.2.2 Architecture Models

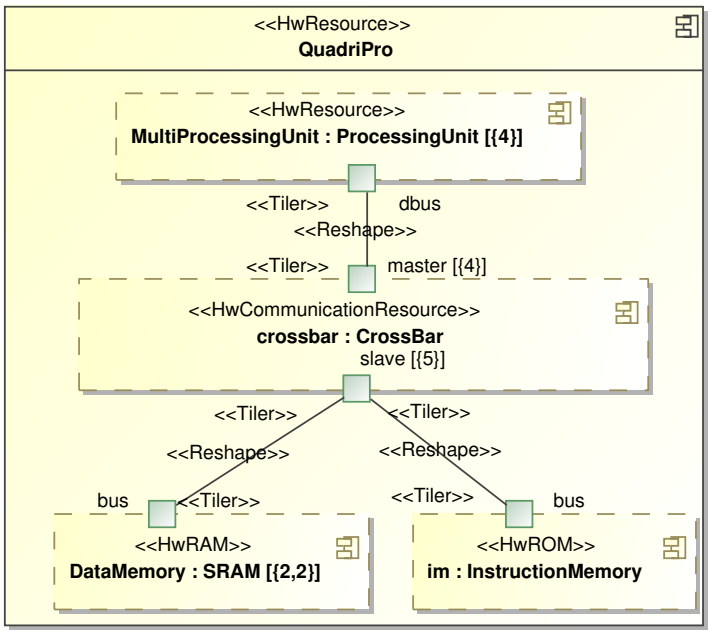

Figure 10: Architecture of a shared memory MPSoC.

Fig. 10 shows a model of an MPSoC architecture. The main component of this architecture, called QuadriPro, is composed of four processing units specified using the repetition concept, four RAM modules specified as a 2×2-grid, a ROM, and a crossbar that interconnects these components together. The Reshape connector (whose attributes are not displayed in this figure for the sake of simplicity) between the dbus port of MultiProcessingUnit and the master port of the crossbar specifies how processors are connected to the communication network. Similarly, the Reshape connectors relating the slave port of the crossbar to the bus ports of memories specify how the repetition of the ports are linked to the ports of the memories.

Now, let us consider another architecture model as described in Fig. 11. One important difference between this architecture and the previous one is that it assumes a *distributed memory* instead of shared memory. This architecture consists of a toroidal 3×3-grid of processing nodes where each node holds its own memory. The two InterRepetition connectors link the nodes together, so that each node can communicate with its four neighbours: north, south, west, and east adjacent nodes.

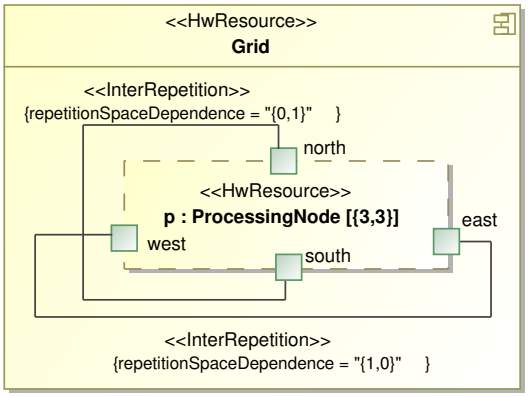

Figure 11: Architecture of an MPSoC with distributed memory.

5.3 Allocation

5.3.1 Mapping Concepts

Since hardware architecture and application functionality are independently modeled, the *allocation* (also referred to as *association*) is in charge of expressing the mapping of the latter onto the former. More precisely, the Allocate stereotype provided in MARTE allows one to specify the mapping of the computations on processing resources and the mapping of data on memory units. In order to define allocation on models containing repetitive structures, a notion of *distribution* is also provided in MARTE. The Distribute stereotype proposes a way to express regular distributions from an array of task (resp. data) to an array of processors (resp. memory units). More details on its semantics are given in [13].

5.3.2 Application Mapping on Architecture

First scenario The components on the top and bottom of Fig. 12 correspond to two different hierarchical levels of the H.263 encoder application model. In the middle of Fig. 12, the main component of a hardware architecture, as seen previously in Fig. 10, is shown. The whole H263mb task is mapped on the processors of the QuadriPro hardware architecture via the Allocate links specified in Fig. 12. The Distribute stereotype (whose attributes are not shown in the figure) specifies precisely the distribution of the 99 repetitions of H263mb onto the four processors. Informally explained, the distribution linearizes the 11×9 repetitions and maps them on the array of the processors via a modulo function. Consequently, 25 repetitions out of the 99 ones are assigned to each processor, except the fourth processor which receives only 24 repetitions.

Similarly, all the data handled in the H263Encoder application are mapped onto the DataMemory. Let us notice that, here, all the memory units are accessible from all processing units, *i.e.* the modeled architecture is shared memory. Therefore the distribution does not require to closely fit the distribution of the tasks. A simple one, allocating a quarter of each data array to each memory unit, is selected. Moreover, the data array allocation can be more precise. For example, the luminance and the chrominance parts of the overall frames can be distributed independently by directly allocating the lumin, cbin and crin ports of the H263mb task onto memories.

Second scenario Fig. 13 illustrates another association where the H.263 encoder application is executed onto an architecture including processors and the acc:Accelerator hardware accelerator (see also Section 8). Compared to the first scenario, this association specifies that the dct:DCT task is mapped onto acc:Accelerator. Indeed, the Allocate dependency starting from the dct:DCT task specifies that the accelerator executes this task. This association leads to a hardware execution of the whole dct:DCT task while the H263mb:H263MacroBlock task is managed in software: for each repetition of this hierarchical task, a processor launches an execution on the hardware accelerator in order to perform the dct task. The quant and the hc task are still mapped onto the processors.

The modifications introduced in this second scenario aim to explore the design space solutions in order to increase the performances of the expected embedded system. Such an exploration starts with the introduction of a hardware accelerator in the architecture model and with a modification of the task allocation in order to exploit the complementarity of both hardware and software executions (see Section 8 for details). Further scenarios could be also imagined. Indeed, according to the SoC designer objectives, the number of processors can be modified, the communication resources can be changed, additional memory units can be considered, etc. The exploration of such a large design space is facilitated at a high abstraction level since it offers the opportunity to observe the overall embedded system without huge implementation details of low abstraction levels. The generation of these implementations requires additional information, which are provided by the deployment mechanism.

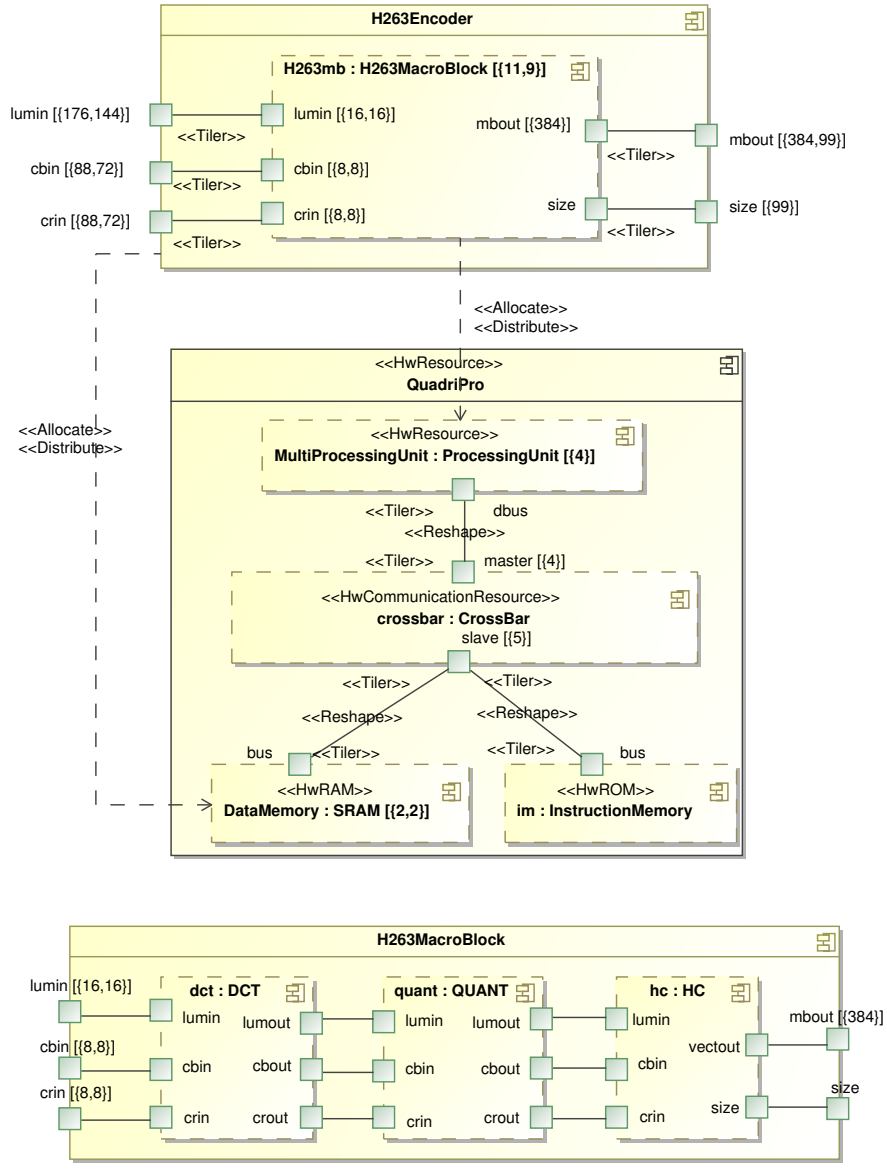

Figure 12: Mapping onto a processors based architecture.

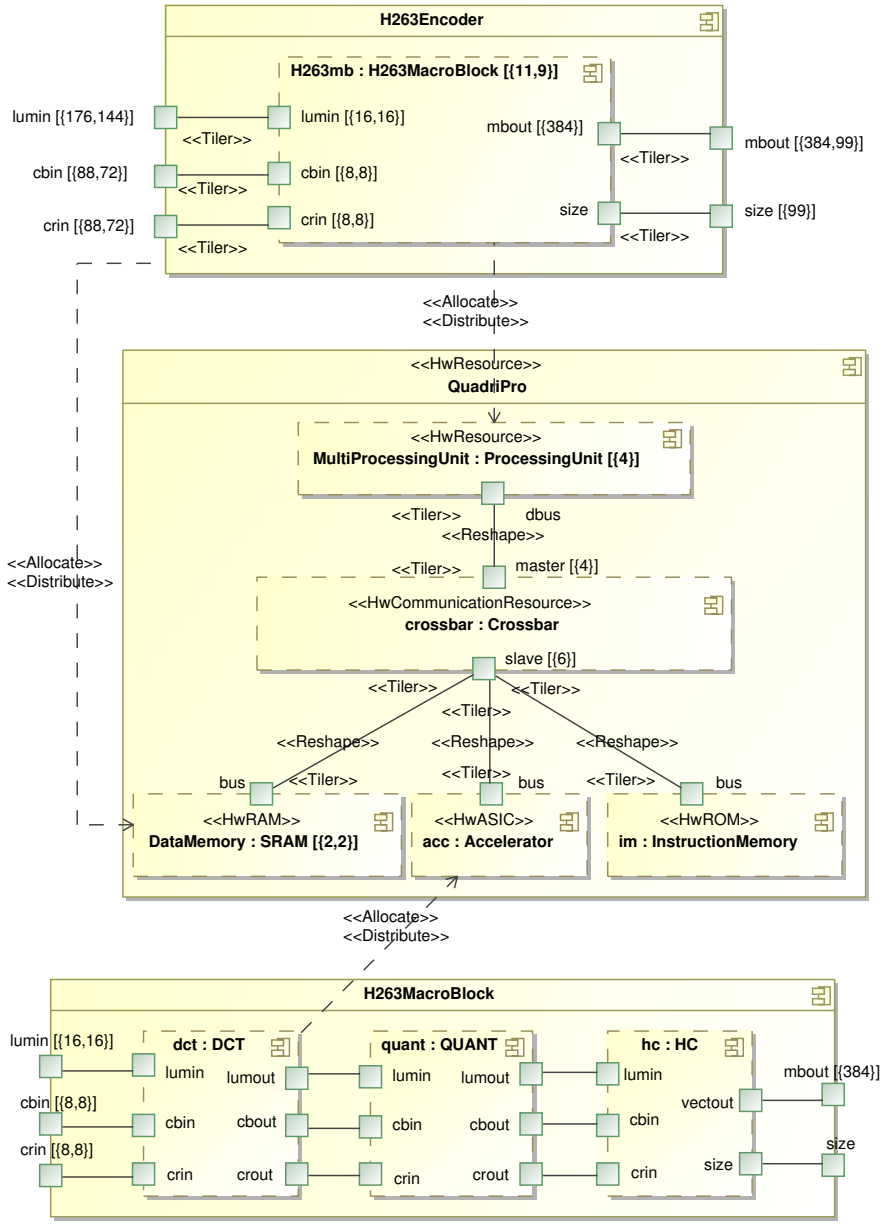

Figure 13: Mapping onto an architecture including a hardware accelerator.

5.4 Deployment using IPs

5.4.1 Deployment Concepts

In order to generate an entire system from a high level specification, all implementation details of every elementary component have to be precised. Low level details are much better described by using usual programming languages instead of graphical UML models. In the SoC industry, individual components are called IP (*Intellectual Property*). IPs are also used to ease component reuse. They correspond to one specific implementation of a given functionality, either hardware or software. In SoC design, one functionality can be implemented in different ways. This is necessary for testing the system with different tools, or at different abstraction levels. For instance, different IPs can be provided for a given application component and may correspond to an optimized version for a specific processor or a version compliant with a given language.

Although the notion of deployment is present in UML, the SoC design has special needs, not fulfilled by this notion. Hence, GASPARD extends the MARTE profile to allow deploying elementary components with IPs. For this purpose, we have introduced the concept of VirtualIP to express the behavior of a given elementary component (either software or hardware), independently from the usage context. A VirtualIP is implemented by one or several IPs, each one being used to define a specific implementation at a given abstraction level and in a given language. Finally, the concept of CodeFile is used to specify, for a given IP, the file corresponding to the source code and its required compilation options. The used IP is selected by the SoC designer by linking it to the elementary component through the Implements dependency. Some IPs provided by the SoC industry can be parametrized. These parameters are specified using the Characteristic concept.

5.4.2 Deployment Example

Fig. 14 represents the deployment of the Elementary-DCT elementary component onto the VirtualDCT VirtualIP. The VirtualDCT may be implemented by two different SoftwareIP. The DCT-VHDL IP corresponds to VHDL code for an implementation at the RTL level. The DCT-C IP corresponds to C code that is executed on a processor. A CodeFile is associated with each of these two IPs. Thus, designers can choose among the two implementation options.

In this figure, the implements dependency from DCT-C to Elementary-DCT specifies that the former is selected as the implementation of the latter. This dependency also allows one to parametrize the IP by setting the size characteristic to 8. Here, 8 corresponds to the array size needed to execute the DCT functionality. The other implements dependencies indicate how the ports of VirtualDCT (resp. DCT-VHDL and DCT-C) are linked to the ports of Elementary-DCT (resp. VirtualDCT).

Such a deployment mechanism allows one to easily modify the selected IP or its corresponding characteristics, without any modification of the application or the architecture.

5.5 Benefits of the MARTE Modeling Concepts

The concepts and illustrations presented in the above sections show the appropriateness of MARTE to model high performance embedded systems. The use

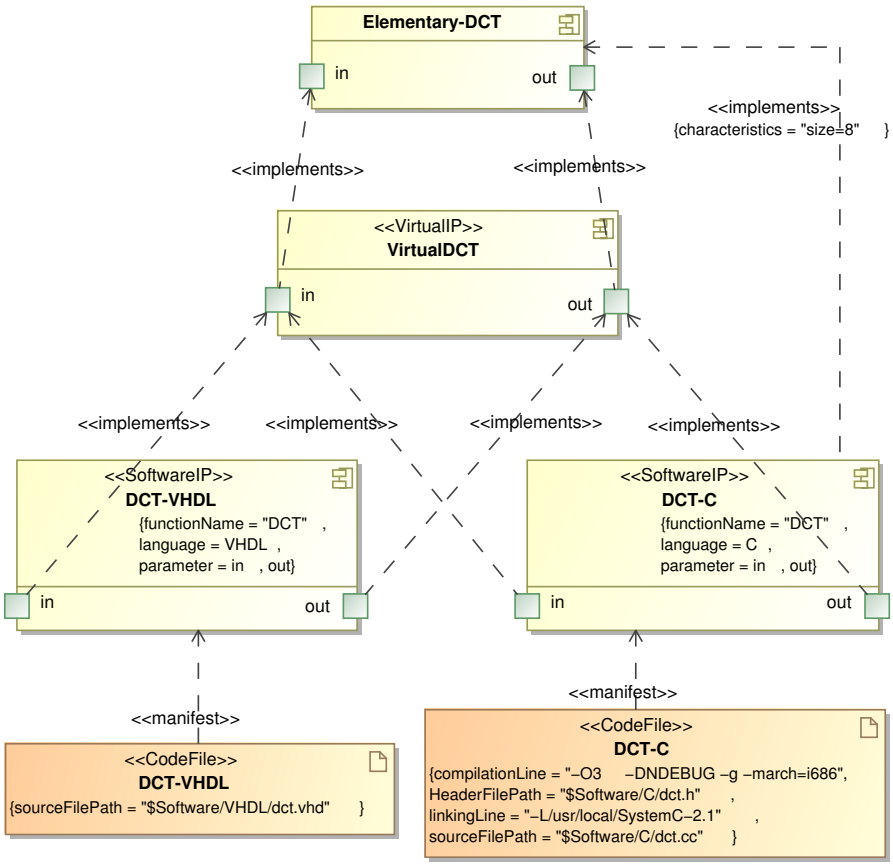

Figure 14: Deployment of the Elementary-DCT task.

of the repetitive MoC enables an *efficient and factorized high level expression* of parallelism that is inherent to a system, in both its application and hardware parts. An interesting point is that the design specification does not suffer from any *scalability* problem that may happen when addressing massively parallel systems. Furthermore, the expressiveness of this MoC allows one to straightforwardly specify different hardware architecture topologies such as the toroidal 3×3-grid of processors or the QuadriPro examples defined in Fig. **??**. This is an important benefit of the MARTE RSM package-based modeling in comparison with closely related languages, *e.g.*, ALPHA [52].

The proposed modeling concepts promote a *separation of concerns* during system design. Different aspects are distinguished: system functionality, hardware architecture, mapping and deployment on target platforms. Contrarily to most platform-based design approaches [45], there is a clear separation between the platform-independent level of design and the deployment level at which the designer chooses a specific implementation platform. This favors more *flexibility* regarding the selection of the suitable implementations. In addition, all design aspects identified here are *uniformly* specified using a unique design model: the MARTE profile.

Finally, through the modeling of the H.363 encoder example, we have shown how already defined concepts are *reused* thanks to the component-based design adopted in GASPARD. This is very important in order to meet the stringent time-to-market constraints imposed to designers.

6 Model Refinement

We now survey the refinement of the previous high level models, via the transformation chains implemented in GASPARD. Fig. 15 summarizes these chains. The top of this figure corresponds to the high level modeling concepts described in the MARTE profile: Application, Architecture, Association and Deployment. However, the set of common concepts present in the MARTE profile is not explicit enough to directly permit the verification of the model, the expression of details in an electronic circuit or the simulation of a SoC. Thus, ad-hoc metamodels have been defined to introduce intermediate representations based on concepts that are closer to technologies. As part of the same effort, for each specific target technology, a transformation chain has been designed in order to automatically derive the target implementation from the designed model. Different transformation chains are defined, which transform high level models towards specific technologies: synchronous languages for formal validation, SystemC for simulation, OpenMP/Fortran for scientific high performance computing and VHDL for hardware synthesis. They are identified in the bottom side of Fig. 15.

6.1 GASPARD Transformation Chains

The first step is the same in each transformation chain of GASPARD. It consists in targeting the Deployed metamodel. This metamodel corresponds to an intrinsic definition of the concepts, identified in the MARTE profile extended with the deployment concepts without taking into account the UML concepts on which they rely. Models that are conform to this metamodel can be modified using refactoring functions [14]. These functions consist of classical loop

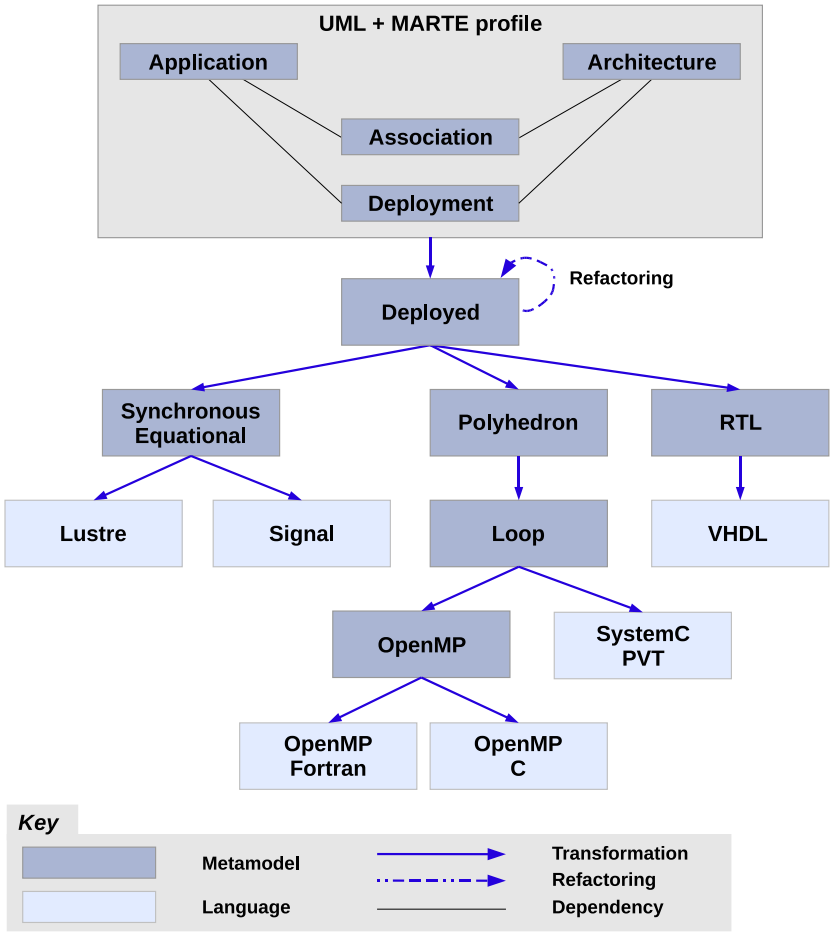

Figure 15: GASPARD transformation chains.

transformations (*i.e.* fusion, collapse, etc.). They modify, create or delete the hierarchy according to the regular parallelism. They can thus be used to adapt the design provided by the user while keeping the functionality unchanged, for example in order to optimize the execution. The other intermediate metamodels are described in the corresponding transformation chains.

6.1.1 Transformation chain Towards Synchronous Languages

The goal of this transformation chain is to provide GASPARD designers with the possibility to address functional correctness issues. Indeed, MARTE does not provide the semantics necessary to formally validate the designed system. Synchronous languages [10] are well-known for their formal aspects and their richness in terms of tools for validation and verification at the functional level. Safe array assignment, causality in data dependencies [23] are examples of functional properties that are checked in an application designed with MARTE.

The *Synchronous equational* metamodel allows a description of applications at the functional level. It relies on the concepts of *Signal*, *Equation* and *Node*. A *Signal* corresponds to a variable. An *Equation* is a relation defining the outputs in term of a function of the inputs. A *node* is a system of equations over signals that specifies relations between values and clocks of the signals. The transformation chain allows the code generation of either Lustre or Signal synchronous language, allowing one to check functional properties of applications described with MARTE.

6.1.2 Transformation Chain Towards SystemC

This chain leads to the generation of a simulator of the embedded system written in SystemC. The targeted abstraction level is PVT (Programmer View Timed). Simulations performed at this level provide information on the embedded system behavior. In addition to the observation of the application behavior, its execution time can be estimated by taking into account the possible data contentions in the communication resources (*e.g.* when several processors try to access the memory at the same time). Furthermore, estimations of the architecture physical properties, such as the energy consumption, can be realized. Since the simulations at the PVT level are fast, many configurations of the embedded system can be evaluated, allowing a wide design space exploration.

The *Polyhedron metamodel* presents the association mechanism from the architecture viewpoint. Instead of having concepts to indicate the placement of tasks and data arrays as in the Deployed metamodel (*i.e.* allocation and distribution), in the Polyhedron metamodel, the data arrays are contained into memories and the tasks are contained in the processors. In order to faithfully represent the repetitions of these distributed elements, the *polyhedron* mathematical concept is used. A polyhedron is basically a set of linear equations and inequalities. In literature, several works on parallel scheduling rely on them because they enable to determine an efficient execution of loop iterations on processors. A whole set of theories and *tools* are available to generate optimized code out of such a representation (our implementation uses the CLooG tool [7]).

The *Loop metamodel* is the second intermediate metamodel. It is very close to the Polyhedron metamodel. The unique difference is the representation of the task repetition. Instead of using a polyhedron, the repetition is represented by a *LoopStatement*. It corresponds to the pseudo-code structure that for a given processor index, goes all over the repetition index of the associated tasks [4]. This metamodel is the last one in the transformation chain towards SystemC. The code is directly generated from this metamodel.

6.1.3 Transformation Chain Towards OpenMP Languages

This chain shares several metamodels with the previous one. However, while the SystemC chain generates code directly from the Loop metamodel, an additional intermediate metamodel is used to generate procedural languages. The OpenMP metamodel is an abstraction of such languages. In this chain, the Single Program Multiple Data (SPMD) execution model is considered: each processor has the same code fragment, parametrized with the processor number.

The *OpenMP metamodel* is inspired by the ANSI C and Fortran grammars and extended by OpenMP statements. The aim of this metamodel is to use the

same model to represent Fortran and C code. Thus, from an OpenMP model, it is possible to generate OpenMP/Fortran or OpenMP/C. The generated code includes parallelism directives and control loops to distribute task repetitions over processors [48].

6.1.4 Transformation Chain Towards VHDL

This transformation chain enables the design of hardware accelerators. A hardware accelerator is an electronic circuit that allows a maximal parallelization of the computation needed to execute an application. It provides an optimal execution support for regular and repetitive tasks. An accelerator is generally dedicated to the execution of a specific application. Each accelerator must be customized separately without real possible reuse. This leads to long production delay and high design cost. Thanks to this chain, it is possible to automatically produce hardware accelerators that solve these two issues and reduce human intervention, which is often error-prone.

The generation of a hardware accelerator first relies on the *hardware-software* partitioning specified in UML. More precisely, it depends on the hierarchical task (application model) that is allocated onto the hardware accelerator (architecture model). This hierarchical task and the lower ones are executed by the hardware accelerator, while the loops contained in the upper hierarchical levels are managed by a processor. The processor iterates on its corresponding repetition space: at each iteration, the processor launches an execution on the hardware accelerator. In this context, the processor is the master and the hardware accelerator is a slave.

In order to generate the hardware design for the tasks allocated on the accelerator, the VHDL transformation chain handles two types of hardware executions for the data parallelism. The *hardware-sequential* execution consumes few resources but provides a relatively long execution time whereas the *hardware-parallel* execution consumes more resources but increase the performances. The partitioning between the hardware-sequential and the hardware-parallel executions are automatically performed in Gaspard [29].

The hardware-software and the parallel-sequential hardware partitionings give the opportunity to customize the performance of an embedded system. Indeed, the design space is large and covers solutions that vary from those which optimize the execution performances to those which minimize the area costs. The exploration in this large design space may be directed by performance and/or area cost requirements.

The *RTL metamodel* gathers the necessary concepts to describe hardware accelerators at the RTL (Register Transfer Level) level, which allows the hardware execution of applications. This metamodel introduces, *e.g.*, the notions of *clock* and *register* in order to manipulate some of the usual hardware design concepts. The RTL metamodel is independent from any Hardware Description Languages (HDL) such as VHDL [25] or Verilog [50]. However, it is precise enough to enable the generation of synthesizable HDL code.

6.2 MDE Benefits for Gaspard

MDE reduces the development and maintainability efforts of high performance computing design tools. At each abstraction level (*i.e.* metamodel), the con-

cepts and the relations between these concepts are precisely defined. This helps to manage the complexity of the tools by dividing the various compilation steps and by formally defining the concepts associated with each step. It also facilitates the development and the extensions of the tools. The transformation chains can benefit from two categories of extensions: *fine grain* and *coarse grain*. A fine grain extension aims to integrate new concepts in metamodels, e.g. in order to extend the supported applications. Indeed, in order to introduce control flow into data flow, some metamodels (*e.g.* Synchronous Equational metamodel) are currently extended. This is successfully realized with the creation of new concepts in metamodels and new rules in model transformations. A coarse grain extension consists of a modification of the design flow itself for new purposes. For instance, one can decide to create a model transformation in order to generate Verilog code from RTL metamodel or to create a RTL model from another metamodel (*e.g.* a metamodel used in another tool). Such flexibilities validate our point of view that advocates the development of tools using MDE: efforts done to develop a tool are capitalized. Therefore, thanks to MDE, GASPARD adapts easily to the changes of the fast evolving SoC domain.

Contrarily to approaches such as UML4SoC [24] or UML4SystemC [42], GASPARD efficiently benefits from the above advantages of MDE. It abstracts more the embedded system specifications and leverages the model transformations to target multiple technologies. Considering the effectiveness of GASPARD for modeling high performance embedded systems, studies [28] have shown that the generated code with GASPARD is reasonably efficient compared to a manually written code. However, a very interesting point is that the automatic generation reduces an important amount of coding time. This is particularly valuable during design space exploration since new implementations are automatically regenerated from simple modifications in the high levels models [9,40]. Such modifications potentially concern the application functionalities, the hardware architecture, the mapping of both or the IP deployment (see Section 8 for more details).

6.3 GASPARD Tool-set

The GASPARD tool-set is provided as an Eclipse plugin [20] that allows users to define embedded systems and to explore the design space based on simulation, synthesis and verification of automatically generated code. The entry point corresponding to the high level specifications is a MARTE-compliant model. Such a model is specified by a user with UML modeling tools such as MagicDraw [35] or Papyrus [39].

The user selects the transformation chain to be executed among the above chains. For instance, the screen-shot of Fig. 16 corresponds to an execution of the transformation chain towards SystemC. The upper panel corresponds to the UML model. The lower panel is the generated SystemC code. The other panels correspond to the intermediate models.

7 Multi-Level Design Space Exploration

A major goal of GASPARD is to rapidly design an embedded system that meets its requirements, in particular those related to performances and correctness.

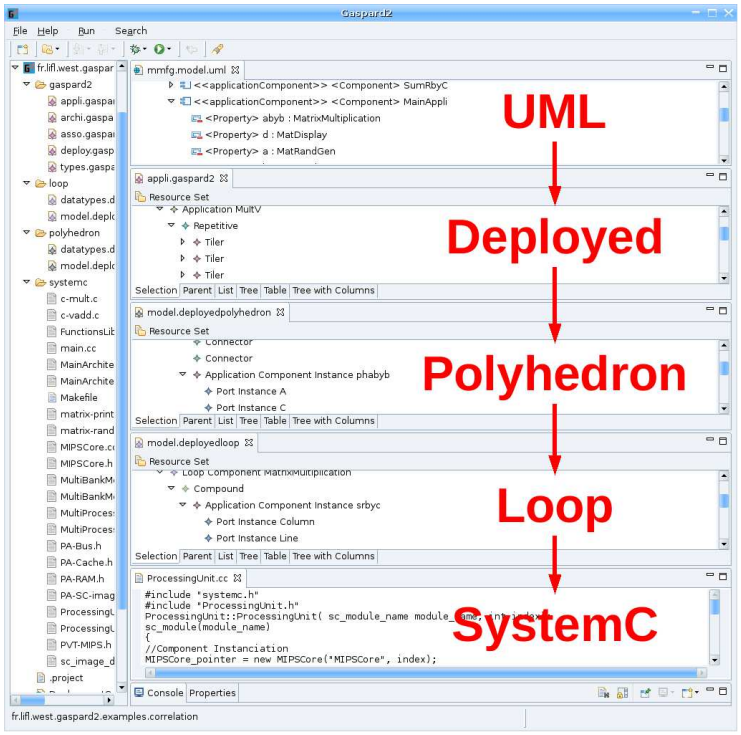

Figure 16: Execution of the transformation chain towards SystemC.

This goal is achieved by considering high level models and the different technologies that are reached via the refinement chains provided in our framework. For that purpose, some precise ordered design steps should be respected. Based on these steps, we define a methodology dedicated to the multi-level design space exploration for high performance embedded systems in GASPARD. This methodology relies on the refinement chains presented in Section 6 except the chain that targets OpenMP languages, which is specifically dedicated to high performance computing for scientific applications.

Given a system to be designed, two main steps are identified: *high level model specification* and *low level analyses*. There are possible feedback loops from the low level analysis to the high level specification in order to enhance the design.

7.1 High Level Model Specification

The designer should exploit the expressive factorization mechanism provided by the RSM package. So, he or she specifies the potential parallelism available in the different system parts: functionality, hardware and the corresponding associations. We distinguish three sub-steps that are combined in different ways.

Description of system functionality and architecture: here, the user defines the application algorithm (*e.g.* the H.263 encoder) on the one hand, and the hardware architecture (*e.g.* a quadri-processor) on the other hand.

Allocation of functionality onto architecture: the user decides a strategy for task and data distribution on the defined hardware architecture. Data are allocated to memory units while tasks are allocated to processing units. Furthermore, a hardware-software partitioning may be specified.

IP Deployment: at this stage, the elementary components used in the system functionality and the architecture are deployed on IPs so as to target a given technology.

While it is obvious that the first sub-step is the starting point of the model specification, the other two sub-steps are applied according to two possible scenarios: either the deployment decisions are partial or complete. The former case enables to postpone a part of these decisions after the allocation of the system functionality on a given architecture. The latter case happens when the user does not necessarily need to wait for allocation to start some design verification or simulation.

Here, we can mention three major advantages in the methodology: an explicit separation of concerns, a use of high level models and a unified specification. These three points ease the reuse of parts in models and the exploration of different system configurations (*e.g.* variation of the number of processors), which significantly reduce the overall design efforts.

7.2 Low Level Analyses

From the high level system models resulting from the previous step, we target three technologies for system analysis at different abstraction levels: functional level with synchronous languages, PVT level with SystemC and RTL level with VHDL. These technologies are used in a complementary way to define a top-down multi-level exploration approach. During each exploration sub-step, the results obtained from the analyses enable to re-design the considered high level models for a better suitability w.r.t the system requirements.

Formal verification of functional properties with synchronous languages. Given a system under design, the synchronous technology is first used to formally check its correctness regarding the functionality requirements. Typically, possible errors in the specification of the algorithm defining the system functionality can be detected. In that case, the user has to go back to the high level specification step in order to correct the embedded system model. Non functional aspects related to architectural details are not addressed in this sub-step. The next two sub-steps are suitable for that purpose.

Simulation based exploration at PVT level with SystemC. Here, the design analysis takes into account both system functionality and architecture. The PVT abstraction level considered in SystemC is sufficiently high to keep the simulation fast enough, and to enable the test of several configurations within a limited time-frame. Such a test includes, *e.g.*, the variation of the number and the type of processors and memory, the modification of the communication network topology, etc. An illustration is given in Section 8 for the implementation of the H.263 encoder application.

Hardware accelerator based exploration at RTL level with VHDL. This sub-step is a suitable complement of the previous one in that it enables to explore further implementation alternatives by taking into account hardware accelerators in the system design. Typically, when the performance requirements of a system are not satisfied during the SystemC-based simulations, one can de-

cide to synthesize some parts of the system functionality as hardware. This significantly ameliorates the execution performances (see Section 8). Among the useful information that are obtained in order to select the best hardware implementation, we mention the execution time and the surface occupied by the accelerator.

The above three target technologies offer a very interesting basis for design space exploration. They contribute to converge efficiently towards embedded systems that meet both correctness and performance requirements.

In the current industrial practice, design space exploration for embedded systems is very expensive to achieve. For instance, the combination of MPSoC and hardware accelerator in the architecture would require two different expert teams. Our design framework aims at carrying out uniformly all the specifications on the same system model using MARTE. Therefore, the design space explorers do not have to deal with the interoperability of several tools, each one being dedicated to a given purpose specification of the system.

7.3 Exploration of Next Generation Embedded Systems

The above methodology relies on the current implementation of the GASPARD framework, and particularly on its transformation chains. It will be very interesting to enhance the complementarity of the transformation chains targeting the PVT and the RTL levels. Indeed, supporting hardware accelerators at the PVT level and processor-based architectures at the RTL level would make possible the simulation of the whole system at each of these levels.

The PVT level enables fast simulations but leads to approximative results. In the opposite, at the RTL level, the results are very precise while the simulation time is very long. The PVT level is thus more adapted to explore large design spaces whereas the RTL level should be used to explore a limited solution set. According to our methodology, the subset of the overall design space determined at the PVT level should be explored at the RTL level. However, this set is still quite large. Thus, its effective exploration at the RTL level is very time consuming due to slow simulations. Henceforth, it is relevant to reduce the design space to be explored at the RTL level. For that, an intermediate subset of the design space resulting from the PVT level has to be introduced. Such a subset may be defined according to the simulations performed at an intermediate abstraction level. Indeed, at the Cycle-Accurate Byte-Accurate (CABA) abstraction level, simulations are performed faster than at the RTL level. The results obtained from these simulations are more precise than those obtained at the PVT level. Hence, the CABA level suitably complements the PVT and the RTL levels. The CABA level can be easily reached in GASPARD by introducing a new transformation chain, which is under development.

By providing a methodology and a tool-set allowing the exploration of a very large design space via successive reductions until reaching a satisfactory solution, GASPARD would successfully face the main challenge of future embedded systems generations that have to perform even more complex applications onto even more potentially powerful architectures. None of existing MDE-based approaches for design exploration [5, 22] offers similar capabilities as GASPARD for high performance embedded systems.

In the next section, we illustrate the exploration of different low level implementations on our running example based on SystemC and VHDL programs

generated from high level specifications defined with the GASPARD modeling concepts.

8 Implementation Results in the Case Study

This section presents an experiment that evaluates the effectiveness of our design methodology and of the overall GASPARD framework. We focus on the design of an embedded system suitable for executing the H.263 video encoder application introduced in Section 3 and modeled in Section 5. We first design a homogeneous processor-based architecture and explore the design space covered by such an architecture. The number of processors is modified in order to increase performance of the embedded system. The performances of different configurations are evaluated according to fast SystemC simulations performed at the PVT level. These evaluations give the performance limits of a processor-based architecture. We thus keep on exploring the design space by introducing a hardware accelerator. We allocate the most time consuming part of the application on this accelerator. Different versions of this accelerator are automatically generated at the RTL level by using the VHDL transformation chain. Finally, the resulting embedded system is heterogeneous since it uses both processors and a hardware accelerator.

8.1 High Level Specification

According to our design space exploration methodology, the starting point is the high level specification of the parts necessary to build the embedded system. The H.263 application model illustrated in Fig. 8 corresponds to the system functionality. The targeted architecture is composed of four processors and a shared memory, as illustrated in Fig. 10. The mapping of the application onto this architecture is the one illustrated in Fig. 12. Finally, the elementary components of the UML model are linked to IPs thanks to the deployment, partially illustrated in Fig. 14.

Once the high level model is specified, the second step of our methodology is performed. It consists of model transformations and low level analyses of the resulting implementation. According to the analysis results, the GASPARD users explore the design space by testing different configurations of the embedded system. So, the users can go back to the specification step, re-execute the transformations and so on. As the model is expressed at a high level of abstraction, the system is very easily modified [4].

8.2 Formal Verification with Synchronous Languages

The H.263 encoder application must be formally verified before starting the design space exploration. For this purpose, Signal or Lustre code is generated by executing the transformation chain towards synchronous languages. With this chain, the data dependencies specified in the high level application model are transformed into equations in the generated code [23]. Usual analysis tools (compilers and model-checkers) dedicated to these languages check whether or not the specification of the H.263 encoder application functionality satisfies some functional properties.

Among these properties, the single assignment and the absence of causality cycle are verified here. The single assignment property ensures that each data of the output frames is computed only once and is not overwritten by another data. The absence of causality cycle ensures that the H.263 encoder is free of any deadlock during its execution.

Since the H.263 encoder verifies these properties, the design space is now explored at the PVT and the RTL abstraction levels successively, as advocated in our methodology.

8.3 Simulation and Evaluation of MPSoC at PVT Level

In this part, we evaluate how the transformation chain towards SystemC simulation of MPSoC at PVT level helps for the early design space exploration. Indeed, this chain allows a fast evaluation of the embedded system: *e.g.*, its execution time, its occupied physical surface, the possible communication contention.

The transformation chain is executed with the 4-processors architecture illustrated in Fig. 10. The resulting SystemC simulation code is executed so as to obtain the execution time. In order to increase the performance (*i.e.* to reduce the execution time), the design space is explored by increasing the number of processors. This consists in modifying two numbers in the high level model specification: the *shape* of the MultiProcessingUnit processor and the *shape* associated to the master port of the crossbar communication resource. Then, the simulation source code is re-generated, and the simulation of the new configuration is executed afterwards.

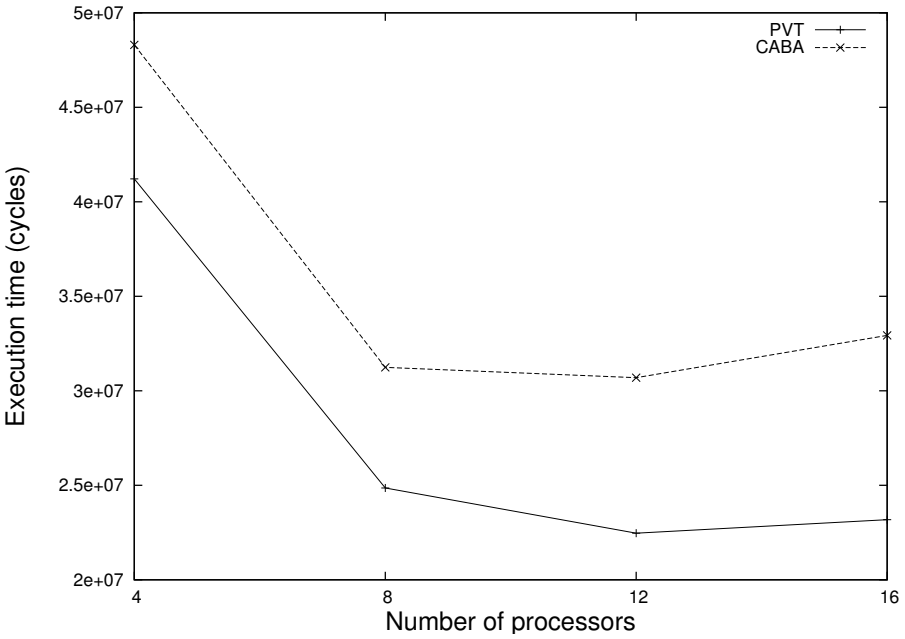

Figure 17: Number of simulated cycles depending on the amount of processors, at the PVT and CABA abstraction levels.

In order to compare the simulation at the PVT level, we have manually written a simulation code of the same system at the CABA abstraction level (Cycle-Accurate Byte-Accurate). The simulation results obtained from this code are supposed to be close to the results that could be observed at the RTL level. Our PVT simulation provides 15% to 30% under-estimated results than this CABA hand-coding, but is faster to execute: in this experiment it was approximately 20 times faster. Besides the CABA slowness, the difficulty to modify the simulation code at this low abstraction level considerably reduces the number of testable configurations in a given time. In addition to accelerate the design space exploration, the PVT code has the advantage to be automatically generated from high level description UML models. This demonstrates the ability to generate fairly relevant simulations of the MPSoC from UML.

Fig. 17 presents the number of simulated cycles used for the encoding of one QCIF frame when the architecture has 4, 8, 12, and 16 processors, at the PVT and CABA abstraction levels. In spite of the under-estimation noted on the PVT simulations, the relative order between the curves resulting from both simulations is coherent. In particular, both simulations show that a 16-processors architecture is slower than a 12-processors one. This is explained by contentions occurring during the data transfer from the shared memory to the multiple processors. The PVT simulation therefore enables us to easily find the optimal configuration (here, 12 processors) for a processor-based architecture.

Based on the methodology, the performance is increased by using a hardware accelerator. Indeed, using such a single but high performance computing resource, the number of memory accesses decreases, and the number of contentions is reduced. The most time consuming part of the H.263 encoder application is the DCT task, which needs up to 92% of the total execution time [8]. This task is known to be efficiently executed with hardware accelerators. In the next part, we address the design exploration by taking into account a heterogeneous architecture.

8.4 Increasing Performance using a Hardware Accelerator

In order to generate a hardware accelerator for the DCT part of the H.263 encoder, the previous UML model is modified. The application part of this UML model remains unchanged. The dct:DCT task is now allocated on a hardware accelerator as illustrated in Fig. 13, and the Elementary-DCT is deployed on the DCT-VHDL IP. For this later, the implements dependency that links the DCT-C IP to the Elementary-DCT task in Fig. 14, is simply moved from the DCT-C to the DCT-VHDL IP. Using the transformation chain towards VHDL, the code corresponding to the hardware accelerator is automatically generated.

8.4.1 Exploring the Design Space

The DCT part of the H.263 application is highly regular and has large repetition spaces in its multiple hierarchical levels. Such large repetition spaces allow us to fully exploit the existing partitioning in VHDL (*i.e* hardware-software and parallel-sequential hardware). This results in several feasible implementations whose performances vary with the chosen partitioning. Fig. 18 represents the characteristics of the generated hardware accelerators for the H.263 encoder DCT part. Each generated hardware accelerator is characterized by an execu-

tion time and an area on the material, necessary for its implementation. In this example, the area is expressed in terms of FPGA resources.

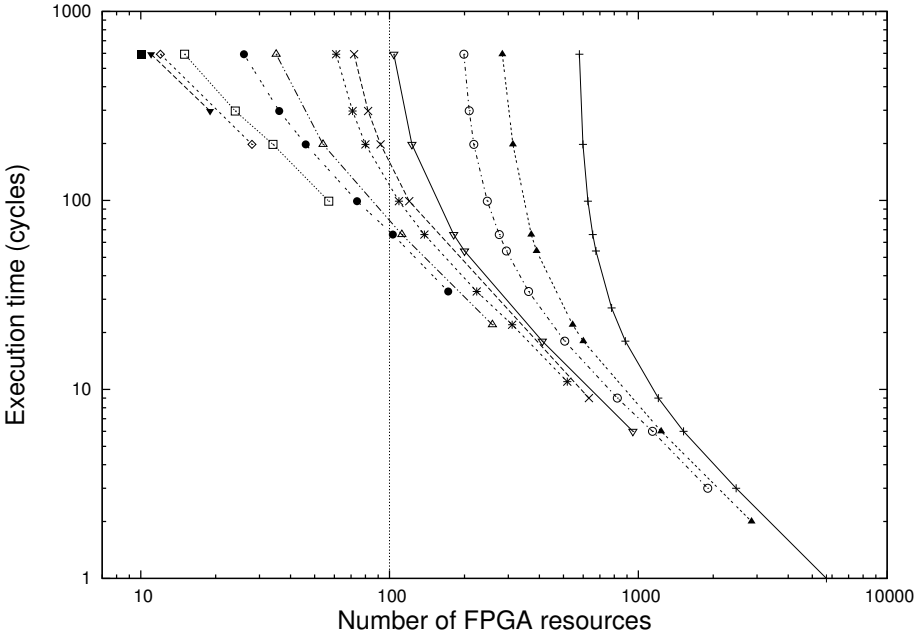

Figure 18: The characteristics of different hardware accelerators automatically generated by GASPARD.

Each curve corresponds to a given hardware-software partitioning, and each point in a curve corresponds to a given sequential-parallel partitioning in the hardware execution. While the hardware-software partitioning is specified by the user in the UML model, the sequential-parallel hardware partitioning is automatically produced. Each point in the figure thus corresponds to a given implementation of the hardware accelerator but takes into account only the execution time of the hardware part. The execution time of the software part is proper to each curve.

The left-most curve is associated with the mapping of the lowest hierarchical level on the hardware accelerator: all the data parallelism is executed in software. The design space is then reduced to a single solution and the resulting embedded system does not benefit from hardware accelerator performances.

In the opposite, the right-most curve is associated with the mapping of the top hierarchical level of the DCT onto the hardware accelerator. The overall DCT is thus executed in hardware (the software part only deals with the management of this accelerator). The performances of the different implementations of the hardware accelerator increase with the number of used resources. However, this number is not realistic compared to the resources provided by nowadays FPGA, even for the most time consuming solution. Indeed, mapping the whole DCT in the hardware accelerator implies that the overall frame is stored on it in order to be managed on one shot. Such a configuration is

thus very resource consuming. The design space is explored between these two extreme hardware-software partitionings. This leads to the intermediate curves.

8.4.2 Selection of the Right Implementation

The selection of the right implementation is driven by the constraints expressed with a maximum execution time and/or a maximum quantity of FPGA resources. When targeting a maximum of 100 resources (illustrated with the vertical line), all the solutions on the left-hand side of this vertical line satisfy this constraint. Among these solutions, the point corresponding to the lowest execution time denotes the most powerful implementation. The VHDL code generated from the right implementation is synthesized with usual synthesis tools.

8.5 Final Embedded System

GASPARD enables us to design an embedded system for an efficient execution of the H.263 encoder application. It includes both general purpose processors and dedicated hardware accelerators. Such a heterogeneous architecture benefits from the complementarity of processors and hardware accelerator: the flexibility of software execution and the effectiveness of hardware execution. Following our methodology, we explored the design space at a high abstraction level using information resulting from the code generated by the transformation chains.

During the exploration, the number of processors has been first modified, then a hardware accelerator has been introduced and, finally, the most effective hardware/software partitioning has been selected. These modifications are easily done in UML, without the inconvenience of low level implementations. Further configurations could be explored, for instance by modifying the number of hardware accelerators, the task allocation, the connexion topology in the hardware architecture, or the deployed IPs. The ease of modifications at high abstraction level coupled with the fast evaluations lead to a very powerful design space exploration framework.

9 Conclusions

In this paper, we have presented the GASPARD framework for the design of high performance embedded systems. It relies on our repetitive Model of Computation (MoC), which offers a powerful and factorized representation of parallelism in both system functionality and architecture. In GASPARD, high level specifications of a system are defined with the MARTE standard profile. The resulting models are automatically refined into low level implementations that are addressed with various technologies for design space exploration: formal verification, simulation and hardware synthesis. These refinements are achieved by using Model Driven Engineering, which enables to clearly define intermediate abstraction levels and to reach them via transformations.

As an answer to the design challenges mentioned in the introduction section, GASPARD offers several advantages: an efficient high level representation of parallelism, a separation of concerns, a reusability, and a unique expressive

modeling formalism. All these features strongly contribute to increase the productivity of designers. We believe that GASPARD adequately responds to the needs of next generation high performance embedded systems.

The whole framework is supported by user-friendly UML editors for system modeling and the Eclipse environment for model refinements. Thanks to the complementary of the different transformation chains that work at different abstraction levels, a user is able to design embedded systems that meet performance requirements as illustrated on the case study.

Currently, the transformation chain towards SystemC only manages software execution on processor-based architecture at the PVT abstraction level, contrarily to the VHDL one which handles hardware execution on hardware accelerator at the RTL level. As future work, we plan to enhance the complementarity of both PVT and RTL abstraction levels in GASPARD by partially merging the corresponding transformation chains. Thus, we should be able to simulate and analyze the overall embedded system at each of both abstraction levels. This enhancement provides additional information on the behavior and the performance. Depending on the design requirements, a designer could focus on the most relevant information for which the exploration could be partially automated using for instance some heuristics.

These evolutions will be integrated in GASPARD using MDE so as to keep on taking advantage of its features. For this purpose, some useful notions to be considered are kinds of traceability in models transformation and composition in metamodels extensions.

Acknowledgments

We would like to gratefully thank all members of the DaRT team at LIFL/INRIA, who actively participate to the development of GASPARD and contributed to the present work.

References

[1] The OpenMP API specification for parallel programming. `http://openmp.org`.

[2] Marcus Alanen, Johan Lilius, Ivan Porres, Dragos Truscan, Ian Oliver, and Kim Sandstrom. Design method support for domain specific soc design. In *MBD-MOMPES '06: Proceedings of the Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD-MOMPES'06)*, pages 25–32, Washington, DC, USA, 2006. IEEE Computer Society.

[3] Eric Allen, David Chase, Joe Hallett, Victor Luchangco, Jan-Willem Maessn, Sukyoung Ryu, Guy L. Steele Jr., and Sam Tobin-Hochstadt. The Fortress Language Specification Version 1.0 Beta. Technical report, Sun Microsystems, Inc., March 2007.

[4] Rabie Ben Atitallah, Eric Piel, Smail Niar, Philippe Marquet, and Jean-Luc Dekeyser. Multilevel MPSoC simulation using an MDE approach. In *IEEE*

International SoC Conference (SoCC 2007), Hsinchu, Taiwan, September 2007.

[5] A. Bakshi, V. K. Prasanna, and A. Ledeczi. Milan: A model based integrated simulation framework for design of embedded systems. *SIGPLAN Not.*, 36(8):82–93, 2001.

[6] Felice Balarin, Yosinori Watanabe, Harry Hsieh, Luciano Lavagno, Claudio Passerone, and Alberto L. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *IEEE Computer*, 36(4):45–52, 2003.

[7] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *13th IEEE International Conference on Parallel Architecture and Compilation Techniques (PACT'04)*, pages 7–16, Juan-les-Pins, september 2004.

[8] A. Ben Atitallah, P. Kadionik, F. Ghozzi, P. Nouel, N. Masmoudi, and H. Levi. Hw/sw codesign of the h. 263 video coder. pages 783–787, May 2006.

[9] Rabie Ben Atitallah. *Modèles et simulation de systèmes sur puce multiprocesseurs – Estimation des performances et de la consommation d'énergie.* Thèse de doctorat (PhD Thesis), Laboratoire d'informatique fondamentale de Lille, Université des sciences et technologies de Lille, France, December 2007.

[10] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. L. Guernic, and R. de Simone. Synchronous Languages Twelve Years Later. *Proceedings of the IEEE*, January 2003.

[11] Dag Björklund and Johan Lilius. From UML Behavioral Descriptions to Efficient Synthesizable VHDL. In *Proceedings of the 20th IEEE Norchip Conference*, November 2002.

[12] Pierre Boulet. Formal semantics of Array-OL, a domain specific language for intensive multidimensional signal processing. Research Report RR-6467, INRIA, March 2008.

[13] Pierre Boulet, Philippe Marquet, Éric Piel, and Julien Taillard. Repetitive Allocation Modeling with MARTE. In *Forum on specification and design languages (FDL'07)*, Barcelona, Spain, September 2007. Invited Paper.

[14] Calin Glitia and Pierre Boulet. High Level Loop Transformations for Multidimensional Signal Processing Embedded Applications. In *International Symposium on Systems, Architectures, MOdeling, and Simulation (SAMOS VIII)*, Samos, Greece, July 2008.

[15] David Callahan, Bradford L. Chamberlain, and Hans P. Zima. The Cascade High Productivity Language. In *9th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, pages 52–60. IEEE Computer Society, April 2004.

[16] J.-P. Calvez. *Embedded Real-Time Systems. A Specification and Design Methodology*. Willey, New York, 1993.

[17] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In *20th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications*, pages 519–538, New York, NY, USA, 2005. ACM Press.

[18] Guy Cote, B. Erol, M. Gallant, and F. Kossentini. H.263+: video coding at low bit rates. *IEEE Trans. On Circuits And Systems For Video Technology*, November 1998.

[19] Frank P. Coyle and Mitchell A. Thornton. From UML to HDL: a model driven architectural approach to hardware-software co-design. *Information Systems: New Generations Conference (ISNG)*, pages 88–93, April 2005.

[20] DaRT Team LIFL/INRIA, Lille, France. Graphical array specification for parallel and distributed computing (GASPARD2). `https://gforge.inria.fr/projects/gaspard2/`, 2008.

[21] A. Demeure and Y. Del Gallo. An array approach for signal processing design. In *Sophia-Antipolis conference on Micro-Electronics (SAME'98), System-on-Chip Session, France*, October 1998.

[22] Francisco Assis M. do Nascimento, Marcio F. S. Oliveira, and Flavio Rech Wagner. Modes: Embedded systems design methodology and tools based on mde. In *MOMPES '07: Proceedings of the Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, pages 67–76, Washington, DC, USA, 2007. IEEE Computer Society.

[23] Abdoulaye Gamatié, Éric Rutten, Huafeng Yu, Pierre Boulet, , and Jean-Luc Dekeyser. Synchronous Modeling and Analysis of Data Intensive Applications. *EURASIP Journal on Embedded Systems*, 2008.

[24] Takashi Hasegawa. An Introduction to the UML for SoC Forum in Japan. *USOC'04@DAC2004, San Diego, California*, June 2004.

[25] IEEE, editor. *Std 1076-1993 IEEE Standard VHDL Language, Reference Manual - Description*. Inst of Elect & Electronic, 1994.

[26] Tero Kangas, Petri Kukkala, Heikki Orsila, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen, Jouni Riihimäki, and Kimmo Kuusilinna. Uml-based multiprocessor soc design framework. *Transactions in Embedded Computing Systems*, 5(2):281–320, 2006.

[27] P. Lanusse, S.Gérard, and F.Terrier. Real-time modeling with UML : The ACCORD approach. In *UML 98 : Beyond the notation*, Mulhouse, France, 1998.

[28] Sébastien Le Beux. *Un flot de conception pour applications de traitement du signal systématique implémentées sur FPGA à base d'Ingénierie Dirigée par les Modèles*. Thèse de doctorat (PhD Thesis), Laboratoire d'informatique fondamentale de Lille, Université des sciences et technologies de Lille, France, December 2007. (In French).

[29] Sébastien Le Beux, Philippe Marquet, and Jean-Luc Dekeyser. A design flow to map parallel applications onto FPGAs. In *17th IEEE International Conference on Field Programmable Logic and Applications, FPL*, Amsterdam, Netherlands, August 2007.

[30] E. Lusk and K. Yelick. Languages for High-Productivity Computing: The DARPA HPCS Language Project. *Parallel Processing Letters*, 17(1):89 – 102, march 2007.

[31] G. Martin, L. Lavagno, and J. Louis-Guerin. Embedded uml: a merger of real-time uml and co-design. pages 23–28, 2001.

[32] Grant Martin and Bill Salefski. Methodology and technology for design of communications and multimedia products via system-level ip integration. In *Proceedings of DATE'98 Designers' Forum*, 1998.

[33] Message Passing Interface Forum. MPI Documents. `http://www.mpi-forum.org/docs/docs.html`.

[34] Kathy Dang Nguyen, Zhenxin Sun, P. S. Thiagarajan, and Weng-Fai Wong. Model-driven SoC design via executable UML to SystemC. In *RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, pages 459–468, Washington, DC, USA, 2004. IEEE Computer Society.

[35] No Magic. MagicDraw. `http://www.magicdraw.com/`, 2007.

[36] Object Management Group. A UML profile for MARTE, 2007. `http://www.omgmarte.org`.

[37] Object Management Group, Inc., editor. *(UML) Profile for Schedulability, Performance, and Time Version 1.1.* `http://www.omg.org/technology/documents/formal/schedulability.htm`, January 2005.

[38] Object Management Group, Inc., editor. *Final Adopted OMG SysML Specification.* `http://www.omg.org/cgi-bin/doc?ptc/06-0504`, May 2006.

[39] Papyrus. Papyrus UML web site, 2007. `http://www.papyrusuml.org/`.

[40] Éric Piel. *Ordonnancement de systèmes parallèles temps-réel, De la modélisation à la mise en œuvre par l'ingénierie dirigée par les modèles.* Thèse de doctorat (PhD Thesis), Laboratoire d'informatique fondamentale de Lille, Université des sciences et technologies de Lille, France, December 2007.

[41] A. D. Pimentel. The artemis workbench for system-level performance evaluation of embedded systems. *International Journal of Embedded Systems*, 1(7), 2005.

[42] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A uml 2.0 profile for systemc: toward high-level soc design. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 138–141, New York, NY, USA, 2005. ACM.

[43] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A model-driven design environment for embedded systems. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 915–918, New York, NY, USA, 2006. ACM.

[44] A. Sangiovanni-Vincentelli. Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[45] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design Test Computers*, 18(6):23–33, 2001.

[46] Douglas C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2):41–47, February 2006.

[47] Bran Selic. Using uml for modeling complex real-time systems. In *LCTES '98: Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*, pages 250–260, London, UK, 1998. Springer-Verlag.

[48] Julien Taillard, Frédéric Guyomarc'h, and Jean-Luc Dekeyser. A Graphical Framework for High Performance Computing using an MDE Approach. In *16th Euromicro International Conference on Parallel, Distributed and network-based Processing*, Toulouse, France, February 2008.

[49] William Thies, Michal Karczmarek, and Saman Amarasinghe. Streamit: A language for streaming applications. In *Compiler Construction : 11th International Conference, CC 2002, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002. Proceedings.*, volume 2304/2002 of *Lecture Notes in Computer Science*, pages 49–84. Springer Berlin / Heidelberg, 2002.

[50] Donald E. Thomas and Philip R. Moorby. *The Verilog Hardware Description Language*. Kluwer Academic Publishers, fourth edition, May 1998.

[51] TILE64 Processor Family. `http://www.tilera.com/products/processors.php`.

[52] Doran K. Wilde. The ALPHA Language. Technical Report 827, IRISA, France, 1994.

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399