Extending MARTE to support the specification and the generation of data-intensive applications for Massively Parallel SoC

M.Ammar, M.Baklouti and M.Abid
CES, National Engineering School of Sfax, Sfax, Tunisia
manel.ammar@ieee.org, mouna.baklouti@ieee.org

***Abstract*—The presented work proposes a MARTE (Modeling and Analysis of Real-Time and Embedded systems) extension for the specification of data-parallel applications designed to be executed on mppSoC, a massively parallel System-on-Chip. These applications can be clearly specified and generated using our transformation chain, which is automated and is a combination of contributions in different domains such as Model-Driven Engineering, MARTE modeling and automatic code generation. The modeling methodology as well as the generation process are validated by an image processing application example.**

I. INTRODUCTION

Nowadays, there is a big development in the SoC domain. Embedded systems, at the present time, are generally dedicated to data-intensive processing applications where huge quantity of data are handled in a regular way by means of repetitive computations. Taking for granted that these applications are going to lead the destiny of embedded systems industry, what do we need to do to support them?

As embedded system design is progressing to adapt software-rich solutions, the appeal of hardware-software co-design for achieving the same design productivity gains that logic synthesis brought to hardware was strong. From a high-level functional model, the detailed and optimized software and hardware implementation could be obtained thanks to the co-design approach [1]. In fact, the incredible evolution in the embedded systems domain could not be achieved by hiring an exponentially growing number of design engineers. It is therefore essential to offset the gradual complexity of SoC design by making the productivity of SoC designers stronger. Adopting new design methodologies and introducing new levels of abstraction allows to strengthen the SoC designers' efficiency. As equipment manufacturers were essential for manufacturing, co-design methodologies are essential for design. These methodologies generally applied notions that raised design abstraction and maintained the modeling precise enough at increasing levels of hierarchy.

When dealing with massive computation and data-intensive processing, the use of massively parallel architectures is very useful. An mppSoC (massively parallel processing SoC) system is a generic massively parallel embedded architecture designed for data-parallel applications [2]. mppSoC is designed based on an assembly of different components and may be implemented on a single chip. In addition, mppSoC proves very fruitful in massively parallel applications domain. However, the design and implementation of such systems become critical due to their long design and development cycles. In fact, the mppSoC design is facing today a strong pressure on reducing time-to-market while the complexity of this system has been increasing. Changing a SoC configuration may also necessitate extensive redesign. Design abstraction offers a possible solution to address the above issues concerning the time-to-market and complexity dilemma.

It is in the context of improving the primary productivity of mppSoC, that our work finds its proper place. One of the primary principles followed during this work is the use of Model Driven Engineering (MDE) for SoC co-design specification and development. MDE is able to benefit from a platform based design approach, allowing abstracting and simplifying the system specifications, at the same time as integrating a compilation chain in order to obtain suitable code from the high level models. The mppSoC co-specification is managed using the MARTE profile that have been offered by the Object Management Group (OMG) [3].

However, while MARTE allows abstract modeling of SoC, in the particular case of mppSoC, the specifications lack suitable semantics for expressing data-intensive computations and parallelism at the high abstraction levels. Our work addresses these limitations and introduces several contributions which help to integrate mppSoC specific application concepts in MARTE. Generating data-parallel application is enabled by raising the abstraction level and providing system-level design automation tools. The intensive parallel application is modeled in an UML (Unified Modeling Language) design environment specifying application and architecture modeling. The mppSoC tool implements the entire system for FPGA (Field Programmable Gate Arrays) prototyping board. It allows the designer to automatically generate the parallel SoC configuration and the data-parallel application, through an exploration step, from high-level system specification models. For this purpose, the mppSoC framework is based on an IP (Intellectual Property) library.

The rest of this paper is organized as follows. An overview of the related works is given in Section 2. Section 3 gives an overview of the data-intensive applications and highlights their specifications. A brief description of the mppSoC architecture will be given in Section 4 focusing on the configurability of this architecture. Section 5 denotes the mppSoC CAD tool that facilitates the co-design of mppSoC through an exploration

step. Our extension for the specification of data-parallel applications will be given in Section 6 and the code generation flow will be well explained in Section 7. Finally, a case study will be given in Section 8 to demonstrate the effectiveness of our work.

II. Related Works

The programming of data-intensive applications for SIMD computers has been widely investigated through the last decades. Research efforts were interested in finding a high level description of the data-parallel applications. These efforts specifically target different SIMD machines and have different levels of abstraction. Two hardware-specific languages were proposed in [4, 5]. CFD and Glypnir were defined to allow the development of a wide range of applications that are designed to be executed on the ILLIAC IV, a SIMD computer for array processing. With the emerging of ICL DAP (Distributed Array Processor) by the start of the 80s, the DAP FORTRAN language [6] was proposed. It basically adds features to the CFD language to support the characteristics of the ICL DAP architecture. Despite its popularity and ability to deal with serious applications, the ICL DAP remains a hardware-specific language. A more recent language that worth to be mention is the *LISP [7] high-level language. It is mainly designed for the Connection Machine CM-2 [8]. While the previously described languages depend on the target hardware requiring the knowledge of their special syntax, this language appeared as compiler directives offering simplified annotations to manage the parallel parts of the programs. Another language designed for the CM-2 is C* [9]. It adds capabilities to the C language aiming to support data-parallel applications. Similarly to the CM-2 machine, the MasPar MP-1 computer was powerful enough that programmers defined a specific data-parallel language aiming to design high-performance programs for this SIMD machine. The language is named MasPar Fortran [10] and is based on Fortran 90. Another Fortran90-based language is the HPF language (High Performance Fortran) [11]. This language supports distributed memory parallel computers.

The previously described languages have shown their ability to successfully design a wide range of data-parallel applications. Thus, they are architecture-specific languages. In fact, all these languages are defined to target one specific SIMD machine. As a result, the grammar of each language depends on the architecture characteristics. In addition, the specification of a data-parallel application remains at an intermediate abstraction level that depends on hand coding. Our goal is to raise the abstraction level beyond these system-level specification languages to be independent from any implementation.

Recently, several design frameworks have been suggested offering high-level system design. The GASPARD [12] framework suggests an MDA-based approach for SoC co-design. It exploits the MARTE profile to model high-performance applications. In [13] a methodology was proposed for the design space exploration of data-parallel applications. It is based on a refactoring tool that manipulates the MARTE-based models to adapt the application's granularity to the architectural and execution constraints. Comparing these related works with our approach, we can observe that none of them supports the complexity involved in the design of data-parallel applications for mppSoC. In fact, this architecture is specific, as a result, the applications designated to be executed on it are also complex. This necessitates a specific tool which takes into account all these characteristics.

III. Data-intensive applications domain

Data-intensive processing is an expanded domain of applications which denotes the manipulation of a considerable amount of data and the accomplishment of numerous complex computations [14]. A data-intensive application is then an application that explores, inquires, examines, pictures and in general deals with very large scale data streams.

Data-intensive applications share many common characteristics which are mainly:

- The high complexity of the data structures

The datasets are in general multi-dimensional. By multidimensional, we mean that they operate multidimensional data structures such as arrays.

- The intensive parallelism available in the application functionalities

The core characteristics of multi-dimensional data processing applications are multi-dimensional data structures that most often express parallel computations. In fact, this data structures are in general intensively repetitive which put on view the native logical parallelism available within such applications. An efficient exploration of the potential parallelism available in the application functionalities is then needed to cope with the applications complexity.

- The high data storage and the computational requirements

Data volumes in data-intensive applications are predicted to grow exponentially. Large data storage is so needed to deal with this big amount of streams. This considerable data streams are often combined with intensive computation to extract useful details. A high performance computation is so required due to the complexity of the data manipulation.

As we have mentioned before, data-intensive applications deal with intensive or massive parallelism. Indeed, parallel applications can implement two levels of parallelism: data parallelism and task parallelism. The data parallelism is performed when all the data are identical computations, in this case data can be processed in parallel. The task parallelism is usually reserved for the situation where we focus on executing different tasks in parallel on different data sets. In our work we are mainly interested by the data parallelism kind.

Data parallelism can be generally defined as a computation applied independently to each of a collection of data, allowing a degree of parallelism that can scale with the amount of data [15]. In fact, data parallelism is commonly turned to account when a computation incorporates some large data-structures that are divided across nodes. Each node accomplishes the same computations on a different part of the data structure.

A data-parallel computation is characterized by a particular data set whose elements have the same basic properties [16]. For the most part, the parallel work concentrates on accomplishing operations on this data set that is normally organized into a common structure, such as an array or cube. A set of tasks cooperatively work on the same data structure, on the other hand, each task manipulates a different partition of the same data structure. These tasks do the same operation on their portion of work. The data-parallel structure of the data-intensive applications exposes the logical parallelism present in the algorithm. An important part of handling the processing needs of these modern applications refers to finding the best execution platform which guarantees an intensive physical data parallelism. The data-parallel behavior of such applications is well realized on SIMD (Single Instruction Multiple Data) architectures. In the next paragraph, we will describe these kinds of architectures.

IV. A TYPICAL ARCHITECTURE FOR THE DATA-INTENSIVE APPLICATIONS

A. *mppSoC general overview*

mppSoC is an IP-based massively parallel architecture [17]. As presented in the figure 1, it is composed of a number of processing elements (the PEs) working in perfect synchronization. A small amount of local and private memory is attached to each PE. Every PE is potentially connected to its neighbors via a regular network. The whole system is controlled by an Array Controller Unit (ACU). Furthermore, each PE is connected to an entry of mpNoC, a massively parallel Network-on-Chip that potentially connects each PE to one another, performing efficient irregular communications.

B. *A parametric architecture for data-parallel applications*

The mppSoC system can be customized to target diverse applications [18]. Our design approach aims to define an mppSoC configuration adapted to a given application. This customization is achieved with the parameterization as well as the extensibility and the configurability of the architecture. In fact, mppSoC is parametric in terms of:

- The number of PEs
- The memories' sizes
- The type of PE: miniMIPS, NIOS, OpenRISC...

It has three configurable aspects:

- The processor design methodology

The processor design methodology is the manner to assemble processor IPs to build the SIMD system. We distinguish two methodologies: processor reduction and processor replication. The former consists on reducing an available open-core processor in order to build a processing element with a small reduced size. This methodology allows putting a large number of PEs on a single chip. Whereas the replication methodology consists on implementing the ACU as well as the PE by the same processor IP so that the designing process is faster.

- The integrated neighboring network's topology

The configurable neighborhood interconnection network is implemented to assure inter-PE communications depending on its configurable topology that can be Mesh, Torus, Xnet, Linear array or Ring.

- The mpNoC interconnection network's type

The mpNoC is integrated to manage point-to-point communications through different types of connections. In fact, the mpNoC includes a configurable router which can be of different types (Shared Bus, Crossbar, Delta MIN (omega, baseline and butterfly)). Added to that, the mpNoC can perform different communication modes (PE-PE, PE-ACU, PE-Device).

Depending on the data-parallel application, we can choose to integrate none, one or both mppSoC networks to build a given mppSoC configuration. We can also vary the design methodology and the values of PE number and memories' sizes to obtain good performances. We clearly notice that there is a compromise between the mppSoC performances and the chosen mppSoC configuration for a given data-parallel application. So, it is important to propose a solution that can select the suitable configuration according to the application constraints.

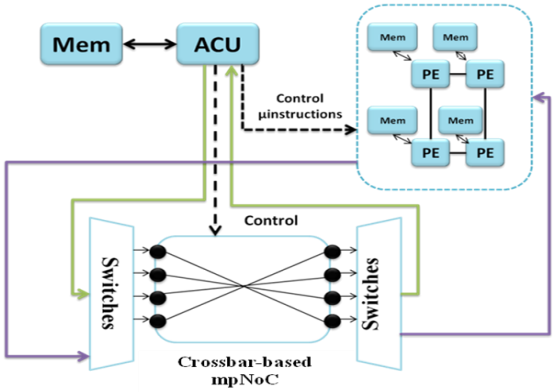

Fig. 1. mppSoC architecture overview

V. THE MPPSOC CAD TOOL

The mppSoC tool is proposed as an Eclipse plugin (figure 2). This tool allows users to specify an mppSoC system, to explore the design space based on high-level analysis, and finally to automatically generate both the application and architecture source 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 Papyrus [19]. A major goal of our tool is to rapidly design an mppSoC system that meets the needed requirements, in particular those related to performances. This goal is achieved by considering high-level models and the different configuration choices 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 high-level design space exploration for high performance mppSoC.

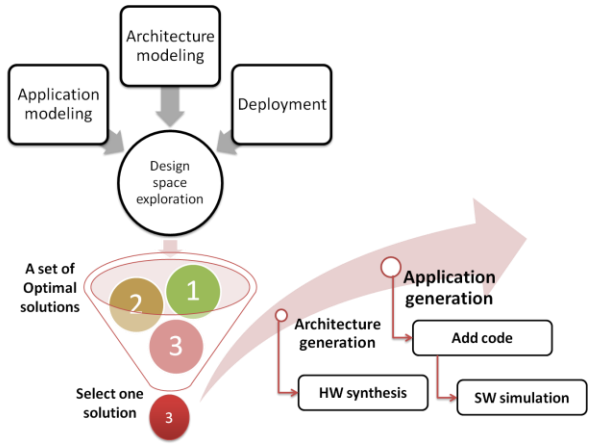

Fig. 2. The mppSoC CAD tool

The designer should exploit the stereotypes provided by the extended version of MARTE. So, he specifies the potential parallelism available in the different system parts: data-parallel application representing the software part and massively parallel architecture characterizing the hardware part. We distinguish four sub-steps that are combined in different ways:

A. *Description of system functionality and architecture*

Here the user defines the application algorithm on the one hand, and the hardware architecture on the other hand. This step depends on the designer choices. If he wants to go through the design space exploration step, he must specify the data-parallel application and model the necessary parts of the architecture. But if he just wants to generate the application and architecture code without exploring design alternatives, he has to specify both the architecture by fixing all parameters related to his configuration. Our tool includes two extensions to model a complete mppSoC system. These extensions allow to model the mppSoC configuration and the data-parallel application. The second extension will be the focus of this paper. General information about the first extension can be found in [20].

B. *IP Deployment*

At this stage, the elementary components used in the system functionality are deployed on IPs. If the designer choose to fix the mppSoC architecture parameters, the deployment of the architecture elementary components becomes necessary. He must specify the mpNoC, the PE and the memories IPs. The deployment phase is ensured using the UML deployment diagram.

C. *Design space exploration*

The design space exploration step relies on two model transformations. These two transformations allow the exploration of a large design space via a multi-objective algorithm. Reaching a satisfactory solution refers to the selection of the appropriate solution. This selection is driven by the constraints fixed by the designer using the user interface.

D. *Application and architecture code generation*

The final step in our co-design flow is the generation of the VHDL (VHSIC hardware description language) code of the mppSoC configuration and the C code of the data-parallel application. For this reason, two chains are defined. The first is the VHDL chain which includes a model-to-text transformation. And the second is the C chain which is based on two transformation types: a model-to-model and a model-to-text transformation.

VI. A MARTE EXTENSION FOR THE SPECIFICATION OF DATA-INTENSIVE APPLICATIONS

To implement a data-parallel application, two main steps are identified: high-level model specification and low-level analysis. The high-level specification of the mppSoC data-parallel application is based on the MARTE profile, the Array-OL language [21] and an mppSoC application profile. The MARTE profile allows the modeling of data-parallel applications in an effective manner, but specific mppSoC details are needed to be represented at a high abstraction level. In fact, mppSoC specific instructions are used to manage communication and control in the architecture. These two mechanisms cannot be modeled based on the MARTE profile. To satisfy mppSoC application needs and to facilitate DSE (Design Space Exploration), we introduce new stereotypes that help the designer to model well structured data-parallel applications.

In order to attempt low-level analysis, a number of transformations are needed. Those transformations generate low-level application code starting from UML input models. In this section, we will focus on the high-level modeling of data-parallel applications that are generated to be implemented on mppSoC configurations. We first explain the capabilities of RSM (Repetitive Structure Modeling) to model the mppSoC repetitive data-parallel application structure. Our extensions for supporting specific characteristics of mppSoC applications are then denoted. These extensions enrich MARTE and allow to obtain well structured models.

A. *Capabilities of RSM for data-parallel application repetitive structure modeling*

Expressing repetitive structures in a clear manner is easy when using high-level abstraction models. In fact, mppSoC data-parallel applications are multidimensional intensive signal processing applications (they manipulate multidimensional data structures such as arrays). The mppSoC architecture is also massively parallel containing multidimensional parallel resources. So the need of a specification language which facilitates dealing with multidimensional structures is primordial. The MARTE RSM package describes the parallel computations in the application SW part and in the architecture HW part in a compact manner. RSM is based on a MoC (Model of Computation) inspired from Array-OL which is a formalism able to represent intensive multidimensional signal processing.

Using the RSM package, an application can be formally specified as a set of tasks. These tasks are connected through their ports that represent information in the form of

multidimensional arrays. These arrays are characterized by their shape, the number of elements on each of their dimensions and their direction. A task can be repeated and each instance of this repeated task operates with sub-arrays of the inputs and outputs of the repetition. The fact that the repetitions of a task are independent, regular and parallel allows expressing data parallelism.

An important stereotype that is present in the RSM package helps to model regular structures. This stereotype, called Shaped allows to model the repetition of a given port or task. The Shaped stereotype allows to specify the shape of multidimensional collection of elements. It then enables to offer a multidimensional view of a collection of elements. Both data and parallel computations of multidimensional structures can be expressed using this stereotype. The shape attribute allows specifying the shape of a collection. The shape includes the number of dimensions and the size of these dimensions.

In a data-parallel application designed to be executed on mppSoC architecture, the shape of the repeated task representing the data-parallel part of the application is related to the PE number. In fact, good performances can be obtained when finding a good mapping of the repeated task on the PE grid. As a result, the shape of this task and the shape of its ports can vary according to the application needs. As the mppSoC architecture is parametric in term of elementary processor number, the amount of PE will be concluded after the design space exploration step. The value of the shape attribute will then remain unspecified which is not a good methodology of modeling. As a result we have chosen to add another attribute to the Shaped stereotype in order to express parametric shapes. This extension is depicted in figure 3. This attribute is called isParametric. It can take two values: true or false, indicating if the shape of the repeated task is parametric or not.

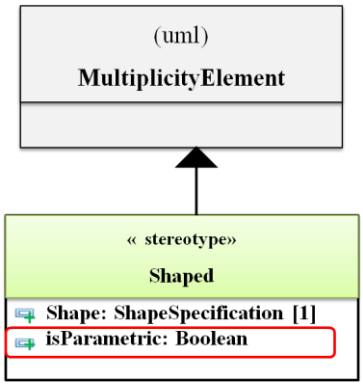

Fig. 3. Extending the Shaped stereotype

B. *Modeling of mppSoC specific instructions*

We identify different instructions to program an mppSoC system: processor instructions, micro-instructions and specific instructions which are encoded from the processor instructions. Examples of specific instructions are:

- Control instructions

These specific instructions control both the regular network and the mpNoC allowing data transfer.

- Communication instructions

Communication instructions are mainly MODE, SEND and RECEIVE instructions.

MODE instructions serve to establish the needed communication mode in the case of mpNoC or the network topology in the case of neighboring communication. After setting the required interconnection, data transfers will occur through SEND and RECEIVE instructions.

SEND instructions serve to send data from the sender to the corresponding receiver.

RECEIVE instructions serve to obtain the transferred data.

To model these mppSoC specific instructions, two stereotypes called CommunicationTask and ControlTask are proposed. The CommunicationTask stereotype allows the modeling of communication between the mppSoC components. The tagged value Type can take two values Load or Store indicating the sending or the reception of data flows respectively. In order to specify the communication mode, a tagged value named Mode is associated to the CommunicationTask stereotype. It can take five values according to the selected mode: 0 (PE-PE), 1 (ACU-PE), 2 (PE-ACU), 3 (PE-Device) or 4 (Device-PE).

C. *Modeling of mppSoC data-parallel application structure*

In a modeled data-parallel application, there are three kinds of tasks: elementary, compound and repetition. An elementary task is a black box. A compound task is a dependence graph, whose nodes are tasks connected via their ports, allowing the expression of task parallelism. Finally, a data-parallel repetition is a repeated task which can be elementary or compound.

The Task stereotype is the main extension introduced in this package. It provides a context for the definition of the basic components of the application model. It extends the SwSchedulableResouce from the MARTE Software Resource Modeling sub-profile, which can describe threads, process, or tasks, to make it more specific for the mppSoC data-parallel applications. WorstExecutionTime and MemorySize tagged values are associated with the Task stereotype to be used in the DSE step. The first is used to specify the maximum time needed for the task execution. And the second expresses the needed instruction memory size occupied by this task.

Two stereotypes CompoundTask and ElementaryTask are used to model elementary and compound tasks respectively. A compound task contains elementary or other compound task(s).

The ElementaryTask stereotype extends the Task stereotype. It is applied to a component which is seen as a black box. It concretely means that this component does not have an internal structure description, or at least that the internal structure description will not be specified in the UML model. During the deployment, an IP will have to be associated with this component to be able to implement the model. The CompoundTask stereotype also extends the Task stereotype. It models a task which encloses elementary tasks and/or other compound tasks.

To facilitate the DSE, it is recommended to model the data-parallel application partition using a specific DataParallelPartition stereotype. This stereotype represents a task or a number of tasks which will be repeated. The Priority tagged value is associated with this stereotype. It is used to indicate the order of the partition execution. It is needed when the application is composed of many data-parallel computations.

VII. A TRANSFORMATION CHAIN TOWARDS C LANGUAGE

In this section, we survey the refinement of the data-parallel application models, via our transformation chain that allows to obtain C code taking as input a model conform to the MARTEDeployed meta-model. An additional intermediate meta-model is used to generate C code, the C meta-model, which is an abstraction of the C language. The transformation chain will be described in the following paragraph.

A. *The C meta-model*

The C meta-model is inspired from the C grammar and extended by mppSoC specific statements. It gathers the necessary concepts to describe a data-parallel application dedicated to the mppSoC system. The developed C meta-model allows generating the global structure of the application and the designer has to complete the missing code.

The C meta-model is explained based on the ECore diagram of the meta-model. We do not express every times detail related to the meta-model, but instead choose to focus only on significant key points. We now describe the necessary details required for expressing a C program.

The main file encapsulates the main function where the program starts execution. Source files contain a number of routines. Header files commonly contain declaration of subroutines, variables, and other identifiers. A C application is mainly composed of three basic components which are the main file, zero or more source files and zero or more header files. These basic components are present in the meta-model by means of Main_File, Header_File and Source_File EClasses.

The statements of a C program control the flow of program execution. In C, as in other programming languages, several kinds of statements are available to perform loops, to select other statements to be executed, and to transfer control. At the present, we only use two statements which are the "for statement" and the "call statement". Other statements that don't belong to the C language are added to express specific mppSoC instructions which are the ControlStatement and the CommunicationStatement.

B. *The MARTE2C transformation*

The first step is the same in each transformation chain. It is based on the UML2MARTE transformation that enables both the design space exploration and the application and architecture code generation. Once the MARTE model has been created from this transformation, we move into the second model transformation in our design flow. The MARTE2C transformation takes the MARTE model as input and generates the C model.

With regards to the C model, we are going to present some significant rules: A CompounTask is transformed into a Source_File. The same CompounTask is converted into a Header_File component. An ElementaryTask must be deployed on a source file that contains a procedure having the same name of the file and describing this elementary task. We suppose that for each ElementaryTask we have to associate both a source file and a header file. The source file contains a procedure that has the same name as the StructuredComponent. The header file should contain as a result the declaration of this procedure. Each CompoundTask contains a number of properties which are instances of other CompoundTask or ElementaryTask. As a result, each property is switched into CallStatement if the shape of the property is 1, or into ForStatement if the shape of the property is more than 1. In this case the attribute bound of the ForStatement will take as value the shape of the property.

C. *The C code generation*

The MARTE2C model transformation presented earlier, converts all the modeled concepts present in the high-level models into their near equivalent concepts in the C model. For code generation, acceleo based templates are written to parse through these concepts, in order to get the required information for the generation of the application code.

Initially the main template deals with the Application component of the C model. It calls three other templates that allow to generate code for the Source_File, the Header_File and the Main_File components using the following code:

```
[template public c_code(source : Application)]

[for (sc : Source_File | source.applicationElement
->select(oclIsKindOf(Source_File)))]
[sc.generatesourcefile()/]
[/for]
[for (h : Header_File | source.applicationElement
->select(oclIsKindOf(Header_File)))]
[h.generateheaderfile()/]
[/for]
[for (m : Main_File | source.applicationElement
->select(oclIsKindOf(Main_File)))]
[m.generatemainfile()/]
[/for]
[/template]
```

For each component of type Source_File, a source file will be generated. It contains global variables, headers, and a procedure that calls other procedures. Components of type Header_file will be converted into header files which contain procedure declarations. One Main_File component exists in the C model. It is transformed into a main file and global variables are generated, call statements are also treated in order to generate the appropriate code. Special instructions are also generated in the main file and the source files. Headers necessary to be enclosed with this file are also added.

VIII. CASE STUDY

A simple RGB to CMYK color conversion application is chosen to illustrate the design process with the presented mppSoC design flow. This application, extracted from the

EEMBC (Embedded Microprocessor Benchmark Consortium) benchmark [22], is widely used in color printers. It has been generated and executed on mppSoC configurations using the proposed model-driven based framework.

A. Modeling the data-parallel application

The RGB to CMYK conversion application is composed of two communication tasks to respectively read and write pixels from devices, and one repetitive computation task to perform color conversion. To model the whole application, we have firstly defined the elementary components: the computation task that performs the conversion function: Conversion, and the two communication tasks: Read_pixel, and Write_pixel, and associated to each component a number of ports. Then, two main components are modeled:

- The Elementary_Conversion component (Figure 4): denotes the three steps of the RGB to CMYK color conversion application.

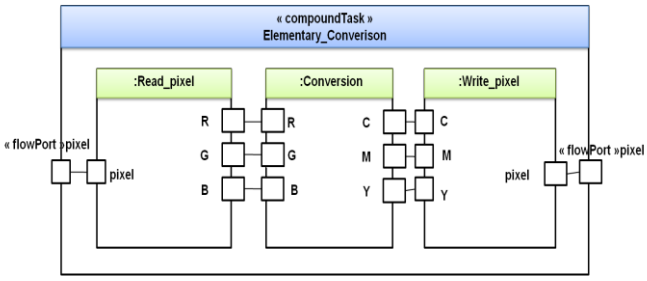

Fig. 4. The MacroBlock component

- The RGB_to_CMYK_color_conversion component (Figure 5): represents the main application and aims to separate the input video frame to blocks. These blocks are read from the SDRAM and sent to all PEs. Then, every block is allocated to one PE and finally the parallel computed blocks are written in an SRAM memory.

B. Modeling the mppSoC configuration

For the tested application, our mppSoC configuration is composed of the ACU, a set of parametric PEs and two I/O devices. The presence of devices necessitates the modeling of the mpNoC. The mppSoC model is described in a composite structure diagram as illustrated in Figure 6.

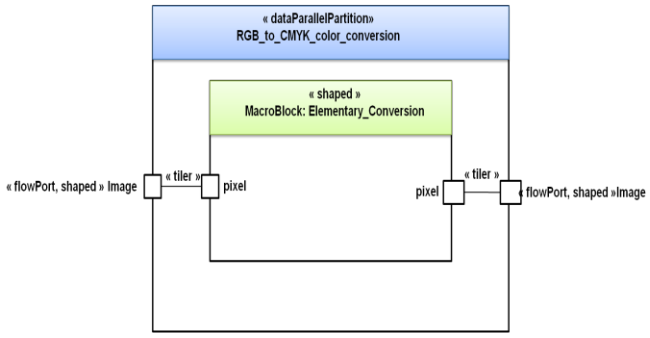

Fig. 5. *The RGB_to_CMYK_color_conversion component*

This diagram models necessary HW components integrated in the system as well as their connections. The memory sizes are not specified. Only three elementary components are modeled: the SDRAM device, the SRAM device and the FPGA.

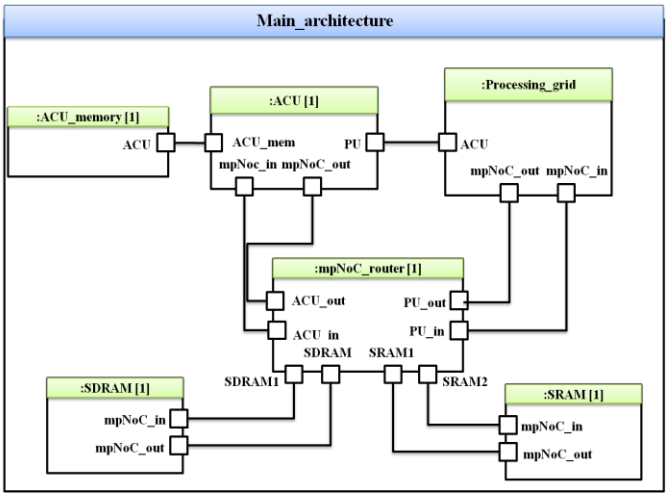

Fig. 6. The main mppSoC architecture

C. IP deployment

To generate the complete hardware implementation, we can choose the devices IPs to be integrated in the final system. The other parameters will be automatically deduced from the design space exploration step. Elementary components of the application must be linked to their implementations. The application elementary components deployment is necessary to elaborate a precise design space exploration.

D. Design space exploration

Given the architecture, application, and deployment models, the design space exploration process takes design decisions about the architecture components parameters. Our tool selects the number of PEs and maps tasks to them in order to minimize the execution time. Repetitive parallel tasks are mapped to a number of PEs and sequential tasks are mapped to the ACU. Tasks execution times are calculated using the specified software IPs from the deployment step. The total execution time is calculated varying the number of PEs for the three processor types available in the mppSoC exploration library. Our tool simultaneously maps processors to a communication structure, such that the overall communication cost is again minimized. A communication structure is needed if there are communications between the architecture components as it is the case of our application.

The exploration tool will decide if the application needs an mpNoC or a regular network to manage the transfers of data flows. This decision is taken after calculating the total communication time needed using different communication structures. A minimal communication time and optimal communication structure surface is needed. Varying the number of PEs allows changing the data memory size needed for each task. This leads to different measurements of the memory size depending on the number and the type of processor. Estimated performance, surface and memory costs

for hardware and software components, provided in the mppSoC exploration library, have been used to guide the exploration. The number of design alternatives in this design space exploration scenario is huge, so that the utilization of an automated design space exploration approach at a very high abstraction level is fully justified. The Pareto-set is finally given resulting from a number of model transformations. The selection of the right solution is driven by the constraints expressed in the user interface (execution time, FPGA area, energy consumption...).

E. Code generation

The low-level synthesizable models from the hardware and software mppSoCLib are used for the final implementation. The right solution is generated, via the exploration step, using the mppSoC flow transformation chains. This solution includes parallel architecture configuration codes and data-parallel application codes.

Our tool enables us to design an mppSoC system for an efficient execution of the RGB to CMYK color conversion application. Following our methodology, we explored the design space at a high abstraction level using information resulting from the models generated by the transformation chains. The ease of modifications at a high abstraction level coupled with the fast evaluations leads to a powerful design space exploration framework.

IX. CONCLUSION AND FUTURE WORK

Through the Model Driven Engineering approach presented in this work, a designer can specify his data-parallel application to generate the executable binary code. At the same time, he can describe his needed mppSoC configuration using UML models and the MARTE profile at a high abstraction level and automatically generate its implementation at RT level. The designer can easily and automatically generate the optimal solution satisfying his needs through a high level DSE tool. Future works aim to generate data-parallel application source code from both structure diagrams and behavior diagrams. While structure diagrams were of our primary interest, behavior diagrams allow the generation of the whole application as the behavior of each task can be modeled in a detailed manner. As a result new transformation rules need to be defined and new meta-models need to be developed.

REFERENCES

[1] F. Hannig, H. Dutta, A. Kupriyanov, J. Teich, R. Schaer, S. Siegel, R. Merker, R. Keryell, B. Pottier and O. Sentieys, "Co-Design of Massively Parallel Embedded Processor Architectures," in *Proc. of the 1st Reconfigurable Communication-centric Systems on Chip workshop, ReCoSoC'05*, Montpellier, France, June 2005.

[2] M. Baklouti, "A rapid design method of a massively parallel System on Chip: from modeling to FPGA implementation". Available: http://tel.archives-ouvertes.fr/tel-00527894/en/.

[3] Object Management Group. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, version 1.0. Available: http://www.omg.org/ spec/ MARTE/ 1.0/PDF/.

[4] K.G. Stevens, Jr., "CFD -- A FORTRAN-like Language for the ILLIAC IV," *ACM SIGPLAN Notices*, vol. 10, no. 3, March 1975, pp. 72-76.

[5] D.H. Lawrie, T. Layman, D. Baer, and J.M. Randal, "Glypnir-A Programming Language for Illiac IV," *presented at Commun. ACM*, 1975, pp.157-164.

[6] P.M. Flanders, R.L. Hellier, H.D. Jenkins, C.J. Pavelin, S. van den Berghe, "Efficient high-level programming on the AMT DAP," *Proceedings of the IEEE* , vol.79, no.4, pp.524-536, Apr 1991.

[7] C. Lasser, J. Mincy, J.P. Massar, Thinking Machines Corporation, "The Essential *LISP Manual," TM Corp 1986.

[8] L.W. Tucker, G.G. Robertson, "Architecture and applications of the Connection Machine," *Computer* , vol.21, no.8, pp.26-38, Aug 1988.

[9] J. R. Rose et al., "C*: An Extended C Language for Data Parallel Programming," *in Proceedings of the Second International Conference on Supercomputing*, L. P. Kartashev et al. eds, May 1987, pp 2–16..

[10] MasPar Computer Corporation, MasPar Fortran User Guide, Software Version 1.1, 1991.

[11] H.P. Zima, "High Performance Fortran - History, Status and Future*," in Proc. ISHPC*, 2002.

[12] L. Bondé, C. Dumoulin, and J.-L Dekeyser, "Meta-models and MDA Transformations for Embedded Systems," in *Forum on Design Languages, FDL'04*, Lille, 2004.

[13] C. Glitia et al., "Repetitive model refactoring strategy for the design space exploration of intensive signal processing appli-cations," *J. Syst. Architect*, 2011.

[14] R.T. Kouzes, G.A. Anderson, S.T. Elbert, I. Gorton, and D.K. Gracio, "The Changing Paradigm of Data-Intensive Computing," in *IEEE Computer* , vol.42, no.1, pp.26-34, January 2009.

[15] L.S. Nyland, J.F. Prins, A. Goldberg, and P.H. Mills, "A design methodology for data-parallel applications," in the *IEEE Transactions on Software Engineering*, Vol.26, pp.293 – 314, 2000.

[16] E.A. West, and A.S. Grimshaw, "Braid: integrating task and data parallelism," in the *5th Symposium on the Frontiers of Massively Parallel Computation*, pp.211–219, 1995.

[17] M. Baklouti, Ph. Marquet, M. Abid and J.-L. Dekeyser, "A design and an implementation of a parallel based SIMD architecture for SoC on FPGA," in *DASIP*, Bruxelles, Belgium,2008.

[18] M. Baklouti, Ph. Marquet, M. Abid, and J.-L. Dekeyser, "IP based configurable SIMD massively parallel SoC," in the *PhD Forum of 20th international conference on Field Programmable Logic and Applications, FPL*, August 2010.

[19] Papyrus, http://www.papyrusuml.org, 2012.

[20] M.Ammar, M.Baklouti and M.Abid, "A model driven based CAD tool for mppSoC design," in the *23rd International Conference on Microelectronics, ICM' 2011*, Winner of the CEDA's ENG-OPTIM'Contest, Hammamet, Tunisia, December 2011..

[21] P. Boulet, "Array-OL revisited, multidimensional intensive signal processing specification," Technical Report RR-6113, INRIA, Paris, France, February 2007.

[22] EEMBC. The Embedded Microprocessor Benchmark Consortium. Available: http://www.eembc.org/home.php, 2012.