Repetitive Allocation Modeling with MARTE

Pierre Boulet, Philippe Marquet, Éric Piel, Julien Taillard
INRIA Futurs / LIFL
Email: {boulet,marquet,piel,taillard}@lifl.fr

***Abstract*— With the advent of multi-processor Systems-on-Chip (MpSoC), the need for modeling the distribution of a parallel application onto a parallel hardware architecture is increasing. The recent standard profile for the modeling and analysis of real-time and embedded systems (MARTE) provides a notation for the modeling of regular distributions. This notation allows to distribute computations to processing elements, data to shared or distributed memories, etc. In this paper we will highlight the expressivity of this notation and clarify its usage through examples and comparisons to other distribution notations such as in High Performance Fortran.**

I. Introduction

As the number of transistors available on a chip keeps increasing, the trend to increase the processing power on a chip is to use more and more cores. The only scalable way to do this is to reuse several times the same core and to organize these cores in a regular way. Several of such regular massively parallel architectures already exist [1]–[6].

The usage of massively parallel architectures requires that designers are able to manage task and data distributions over parallel processors and memories. This requirement is shared with the High Performance Computing field. High Performance Fortran [7] (HPF) proposes some techniques to express the distributions. An HPF compilation directive specifies to the compiler a way to map data to processors (and indirectly the nearest memory). This distribution directive allocates each array element to an *owner*. Then the *owner-compute rule* is applied: the *owner* executes each program block which affects its owned elements.

The UML profile for the modeling and analysis of real-time and embedded systems (MARTE [8], [9]) proposes some notations to define repetitive structures. This notation is based on the proposal of Cuccuru et al. [10] and extends the multiplicity mechanism of UML to give a *shape* to the potential instances of a multiplicity element. This notation has been used successfully to model data-parallel applications and repetitive hardware such as grids of processors, multi-bank shared memories or interconnection Networks-on-Chip [11]. The MARTE profile also proposes a notation for the distribution of such data-parallel applications onto such repetitive hardware. We will highlight the expressivity of this notation and clarify its usage through examples and comparisons to other distribution notations such as in High Performance Fortran.

II. Allocation

The MARTE profile enables Systems-on-Chips (SoCs) co-design. Co-design consists in designing the application and the execution platform (hardware) separately at first. Once they are sufficiently developed, a mapping of the application onto the execution platform is performed in order to represent the complete system. The MARTE profile allows to model both applications and execution platforms. A MARTE application element may be any UML element suitable for modeling an application, with structural and behavioral aspects. The structural aspect represents the static organization of the elements, basically expressed by resources. The behavioral aspect corresponds to the evolution of the system with the time, commonly expressed by services. An application element could be a computation, a service, or even a Real-Time Operating System (RTOS) function. A MARTE execution platform is composed of a set of connected resources representing the System-on-Chip hardware architecture. It is composed of « HW_Resource » such as computing resource (processor, hardware accelerator), storage resource (cache, ROM or RAM) and communication devices (Bus, Bridge and I/O devices).

Once the application and the hardware architecture have been modeled, one should specify how the applications will be executed on the execution platform. The mapping of application tasks onto the adapted execution platform is an important point of real-time embedded system design. It has a strong influence on the performance and also plays a role in the power consumption. To be able to chose the optimal mapping, several possibilities may have to be tested and compared.

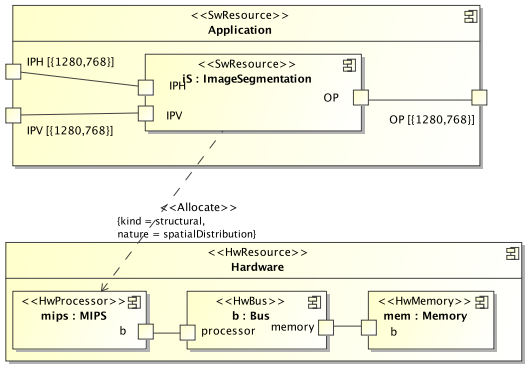

Fig. 1: Allocation of a computation task on a processing resource

In the MARTE profile, the mechanism to specify the mapping is called *allocation*: *a MARTE allocation is an association between a MARTE application and a MARTE execution platform*. The set of all the allocations of the model defines the mapping. The main concept of allocation is « Allocate », it is used *for associating elements from a logical context, application model elements, to named elements described in a*

more physical context, execution platform model elements.

An example of allocation is given in Figure 1. The part *iS*, instance of the *ImageSegmentation* component, is allocated onto an instance of the MIPS processor of the hardware architecture.

Different kinds of allocation are defined: structural, behavioral and hybrid. An allocation of a group of structural application elements on a group of resource is a structural allocation. A behavioral allocation is an allocation of a set of behavioral elements on a service provided by the execution platform. An allocation is called *hybrid* when the application end is behavioral while the execution platform end is structural. This can be used when the execution platform resource provides only one service and this service is implicit, the application service is then allocated on this implicit service.

Allocation can represent either a spatial placement or a temporal placement. Spatial placement is the allocation of computations to processing resources, data to memories at specific ranges of addresses, and of dependencies between SW_Resources to communication resources. Temporal placement is the scheduling of a set of elements spatially allocated to the same platform resource. For instance, several tasks can be scheduled on a processor, or a range of memory addresses can be used at different times to hold different data.

An allocation is valid if both spatial and temporal allocations are consistent. Moreover, there should not be any contradictions between the constraints imposed by the allocation, and the structure and behavior internal to the application or to the execution platform. In particular, allocations must comply with associations and dependencies of each model. For instance, two communicating application elements should not be mapped on two disconnected parts of the execution platform.

In general, there exist potentially plenty of mapping of a given application onto an execution platform. Once a global mapping has been defined, it can be refined in order to find the best allocation of each element, and thus we obtain the best performance and the optimal execution of the application on the hardware.

III. Repetition

The Repetitive Structure Modeling annex of MARTE defines stereotypes and notations to describe in a compact way the regularity of a system's structure or topology. The structures considered are composed of repetitions of structural elements interconnected via a regular connection pattern. It provides the designer a way to express models efficiently and explicitly with a high number of identical components.

A. Shaped multiplicities

When a modeling element has a multiplicity of a fixed integer, n, it can be seen as a collection of n potential elements. With the MARTE profile, these n elements can be organized as a multidimensional array. The shape of this array (number of elements on each dimension) is then specified in the shape tagged value of the « shaped » stereotype. The type of the shape attribute is ShapeSpecification which is a data type corresponding to a vector of unlimited naturals. One can thus consider a collection of 100 components as an array of 5×20 components by specifying a shape of $\{5, 20\}$ using the Value Specification Language of MARTE. For simplicity, instead of using the « shaped » stereotype, the user can write the shape in place of the multiplicity. Whenever a multiplicity is between curly brackets, it has to be understood as a shape specification. Using this notation, one can give the shape of data arrays, of repetitions of application components or of hardware components, etc. Figure 6 shows the use of shape to express an execution platform made of a grid of 16×16 processing unit with only one component to represent the 256 processing units.

B. Repetitive connectors

The second aspect of the RSM package concerns a way to add topological information on relations expressed between design-time entities in order to specify the topologies of links that will exist between run-time entities in the context of these relations.

The design idea is to identify sub-arrays, called *patterns*, of points inside each array (defined by a shape specification), and then to relate the points (i.e. link ends) contained in these patterns. The considered patterns are multidimensional arrays, and thus they are described by a shape similarly to the other arrays. We call *tile* a pattern when it is considered as a set of points of an array. The considered tiles are sets of regularly spaced points and the tiles themselves are regularly spaced in the array. The description of the regular spacing of the points of a tile is called *fitting* and the description of the regular spacing of the tiles in the array is called *paving*. The complete description of the tiling of an array by tiles necessitates the description of the shape of the pattern, the fitting, the paving, an origin and a *repetition space*. The repetition space gives the number of tiles. It is itself characterized by a shape. An example is given in Figure 2, a tile corresponding to a pattern of 3×2 elements is positioned on an array of 6×4 elements.

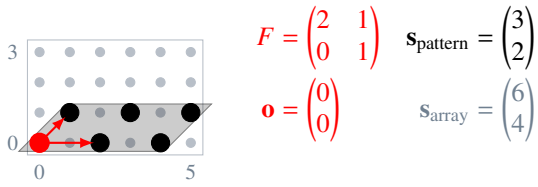

Fig. 2: A sparse tile aligned on the abscissa and shifted by one on the ordinate of the array.

The fitting describes the coordinates of the points of the tile in the array relatively to a reference point. The paving describes the set of reference points of the tiles relatively to the origin. So the origin is the point of index $[0, \ldots, 0]$ of the tile of index $[0, \ldots, 0]$ in the repetition space. The tiling process is described by a TilerSpecification having three attributes: origin, a Vector of Integers, fitting, a Matrix of Integers and paving, a Matrix of Integers. The points of the tile of index $\mathbf{r}$ in the repetition space are enumerated as follows: Given the point of index $\mathbf{i}$ in the pattern, the coordinates of the corresponding

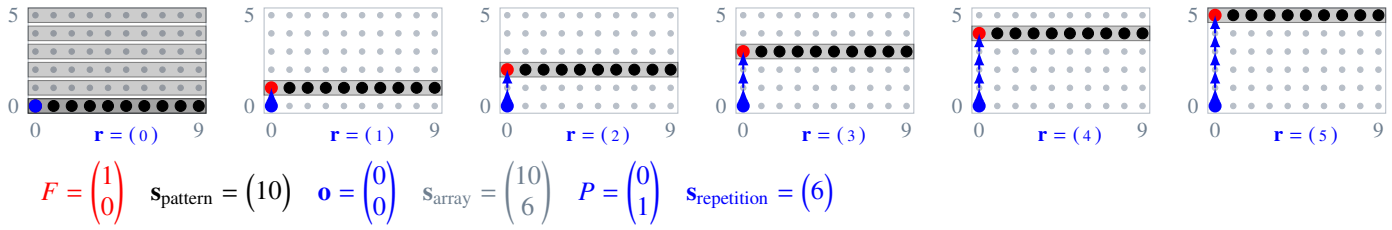

Fig. 3: This figure represents the tiles for all the repetitions in the repetition space, indexed by **r**. The paving vectors drawn from the origin **o** indicate how the coordinates of the reference element of the current tile are computed. Here the array is tiled row by row.

point in the array is

$$\text{origin} + \begin{pmatrix}\text{paving} & \text{fitting}\end{pmatrix} \times \begin{pmatrix}\mathbf{r}\\ \mathbf{i}\end{pmatrix} \bmod \text{array\_shape}\,. \qquad (1)$$

This formula ensures that:

- the points of the tile are regularly spaced because they are built from the reference point of the tile by the linear combination of the column vectors of the fitting matrix,
- the reference points of the tiles are regularly spaced because they are built by the linear combination of the column vectors of the paving matrix,
- all the points of the tiles are points of the array thanks to the computation modulo the shape of the array.

This is strongly inspired by the Array-OL language [12]. The usage of paving to browse the whole array using a tile is shown in Figure 3.

Several stereotypes using this idea are proposed in MARTE, all of them are specializations of the «*LinkTopology*» abstract stereotype: «Tiler», «Reshape», «InterRepetition» and «DefaultLink». The purpose of this paper is not to describe precisely the semantics of these stereotypes, it is done in the MARTE standard and in [12] for the underlying language, Array-OL. We will describe below the formal semantics of the «Distribute» stereotype which is identical to that of the «Reshape» stereotype.

IV. Distribution

A «Distribute» stereotyped link proposes a way to express regular (e.g. block, cyclic, k-cyclic) distributions from an array of elements to another array of elements. It is an extension of the allocation mechanism described above designed to handle the repetitive structures.

A. Semantics

A «Distribute» stereotype specification has to define the following tagged values: *patternShape*, *repetitionSpace*, *fromTiler* and *toTiler*. PatternShape is a ShapeSpecification defining the shape of a pattern that will be used to tile both linked arrays of elements. The repetitionSpace is a ShapeSpecification that defines the shape of the collection of links between tiles defined by the distribution. The fromTiler and toTiler TilerSpecifications define how the source and destination arrays of elements of the link are tiled.

Let us give some notations to all these data. Let $\mathbf{s}_p$ be the pattern shape; $\mathbf{s}_R$ the repetition space shape; $\mathbf{s}_f$ and $\mathbf{s}_t$ the shapes of the *from* and *to* arrays; $(\mathbf{o}_f, F_f, P_f)$ and $(\mathbf{o}_t, F_t, P_t)$ the from and to tiler specifications. The potential link instances can be seen as a mathematical relation, $\rightarrow$, between the elements of the from array, $\mathcal{F}$, and those of the to array, $\mathcal{T}$. This relation is defined as follows.

$$\forall \mathbf{e}_f \in \mathbb{N}^{\dim(\mathbf{s}_f)}, \mathbf{0} \le \mathbf{e}_f < \mathbf{s}_f, \forall \mathbf{e}_t \in \mathbb{N}^{\dim(\mathbf{s}_t)}, \mathbf{0} \le \mathbf{e}_t < \mathbf{s}_t,$$
$$\mathcal{F}[\mathbf{e}_f] \rightarrow \mathcal{T}[\mathbf{e}_t] \Leftrightarrow \begin{cases} \exists \mathbf{r} \in \mathbb{N}^{\dim(\mathbf{s}_R)}, \mathbf{0} \le \mathbf{r} < \mathbf{s}_R, \\ \exists \mathbf{i} \in \mathbb{N}^{\dim(\mathbf{s}_p)}, \mathbf{0} \le \mathbf{i} < \mathbf{s}_p, \\ \mathbf{e}_f = \mathbf{o}_t + \begin{pmatrix}\mathbf{r} & \mathbf{i}\end{pmatrix} \times \begin{pmatrix}P_f\\F_f\end{pmatrix} \bmod \mathbf{s}_f, \\ \mathbf{e}_t = \mathbf{o}_t + \begin{pmatrix}\mathbf{r} & \mathbf{i}\end{pmatrix} \times \begin{pmatrix}P_t\\F_t\end{pmatrix} \bmod \mathbf{s}_t, \end{cases} \qquad (2)$$

where $\dim(\mathbf{s})$ is the number of dimensions of vector $\mathbf{s}$ and $\mathcal{A}[\mathbf{i}]$ is the element of index $\mathbf{i}$ of array $\mathcal{A}$.

This process is powerful enough to link each element in the source array to zero, one, or several elements in the destination array and to express all the classical data-parallel distributions (like those of HPF [7] for example, as we show below).

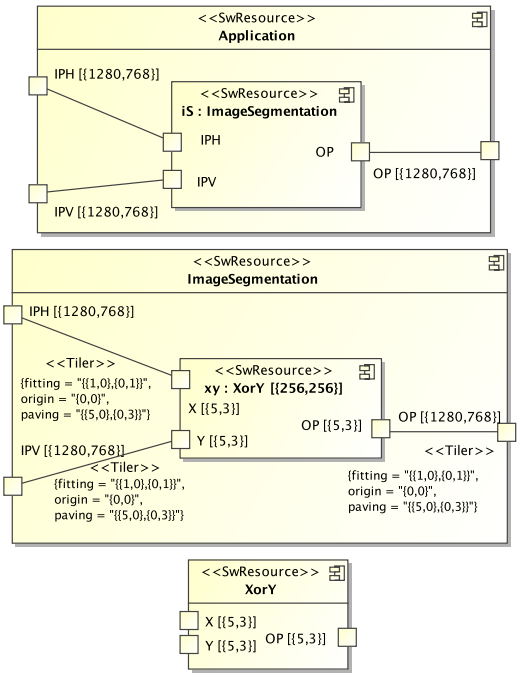

Fig. 4: The application model: a task with three arrays.

B. Examples

In order to clarify and ease the understanding of the semantics description, the use of the « Distribute » stereotype will be illustrated by some examples. The given HPF distributions will be done considering that HPF can distribute tasks in the same way as the data: for simplification, the directive will applied to the *xy* repetitions.

1) Repetitive application: The application modeled in Figure 4 is an extract of a picture analysis application. Starting from a greyscale picture, the picture analysis application detects the edges, and generates a black and white picture where white pixels indicate an edge. The result can then be used in further picture analysis. The edge detection of a picture is done in two steps. First from the input picture, two similar computations are carried out: one is done following the abscissa, the other one is done following the ordinate. Secondly, the image is obtained with an *OR* between each corresponding pixel of the two produced images. Only this second part is presented here.

This is the same application as in Figure 1. In Figure 1, *ImageSegmentation* was considered as a black-box: a coded function which realizes the computation and which has only its interface described at this high abstraction level. The drawback of this simple view is that it is not possible to allocate the computation onto more than one processor. In Figure 4, after analysis of the code, the ImageSegmentation computation has been decomposed into a repetition of the *xy* instance working on small regular patterns. It leads to a {256,256} repetition of *xy* which consumes two {5,3} patterns and produces a pattern with the same shape. Such a decomposition allows the use of the parallel execution platform in order to increase the performance.

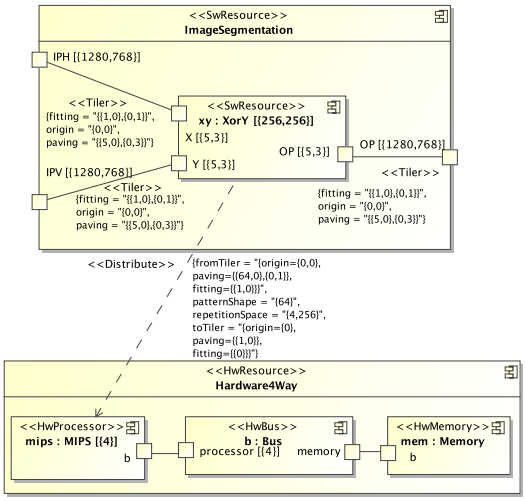

Fig. 5: A distribution of ImageSegmentation on a 4-processor SoC, a block of 64×256 repetitions is assigned to each processor for computing.

2) Distribution on a 4-processor architecture: As a simple example, we extended the execution platform of Figure 1 to contain 4 MIPS processors. To use this introduced parallelism, the allocation has to be converted into a distribution. Figure 5 shows the result, *xy*, the repeated part of *ImageSegmentation*, is distributed on the 4 processors by block. The first dimension of *xy* is divided into 4 blocks, as expressed by the first value of *repetitionSpace*. Each separation contains 64 non-overlapping instances of *xy*, expressed by the *patternShape* and the first value of the *paving* of the *fromTiler*. The second dimension of *xy* is assigned to the processors, as expressed by the second value of the *repetitionSpace* and the second value of the *paving* of the the *toTiler*. The equivalent HPF distribution would look like:

```
!HPF$ PROCESSORS P(4)
!HPF$ DISTRIBUTE xy(*,BLOCK) ONTO P
```

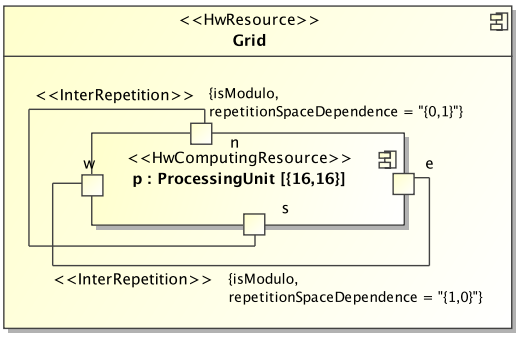

Fig. 6: Grid architecture of the execution platform

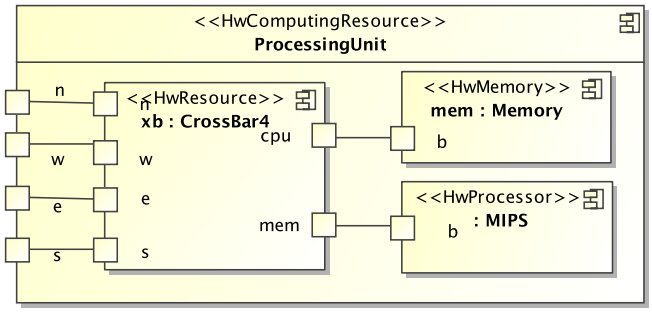

Fig. 7: A processing unit of the grid

Let us note that other ways exist to express the same resulting distribution. For instance, with a *repetitionShape* of {64,256} and a *patternShape* of {4}, we could obtain the same distribution by swapping the first value between paving and fitting of both tilers.

3) Grid execution platform: In order to illustrate other possibilities of « Distribute », a different execution platform is introduced. The targeted architecture is a massively multi-processor System-on-Chip (Figure 6). It is composed of {16,16} repetitions of processing units, which are connected to each other by a toroidal topology. The interconnection topology is modeled thanks to two « InterRepetition » connectors which allow to specify the position of every neighbor of every instance of the *ProcessingUnit* component. In this example, it specifies that each potential instance is connected to its direct neighbors in four directions: north, east, south, and west. It generates a toroidal grid. Each processing unit (Figure 7) is composed of

a MIPS processor, a memory and a crossbar which allows the communication in four directions for each computing unit. In the following, the equivalences given in HPF are distributed onto P defined by the directive:

```
!HPF$ PROCESSORS P(16,16)
```

4) Task distribution: With regard to performance on a parallel system, it is necessary to achieve a good load-balancing of the computation over the processors. In general, the goal is to distribute fairly software repetitions over processors. However, depending on the way the distribution is done, the locality of the data and communications required for the computations will very significantly influence the parallelism speed-up. The optimal distribution of the task depends on the algorithm and on the other distributions of the application. The « Distribute » stereotype permits the designer to specify in detail how the tasks should be allocated on the processors. A task distribution regularly allocates a multidimensional array of tasks onto a multidimensional array of hardware computing resource. The distribution mechanism is illustrated by mapping the repeated *xy* part onto the multi-processor grid in three different ways.

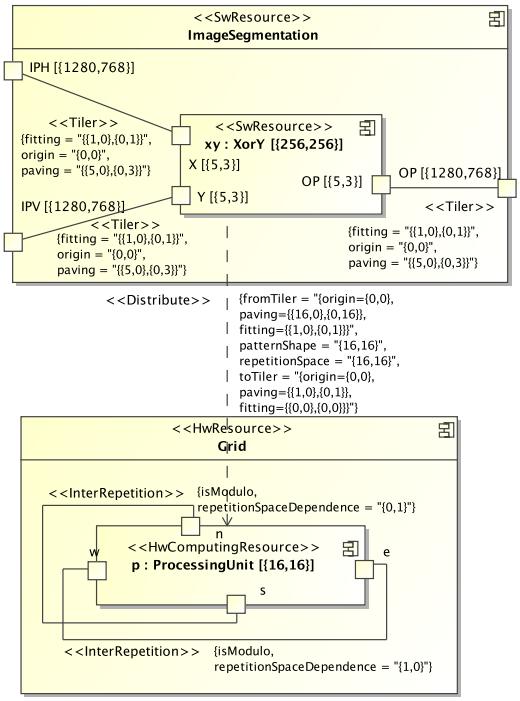

Fig. 8: Block distribution of the {256,256} repetitions of the application part *xy* onto the {16,16} repetitions of the execution platform part *p*.

Firstly, in Figure 8 the repetitions of *xy* are distributed in block along both dimensions. The processor $[i, j]$ will execute the 256 instances from $[i\times16, j\times16]$ to $[i\times16+15, j\times16+15]$. The equivalent HPF syntax would be:

```
!HPF$ DISTRIBUTE xy(BLOCK,BLOCK) ONTO P
```

In Figure 9 the same application is mapped on the same execution platform but following a [cyclic, cyclic] distribution. Each set of 16×16 *xy* instances is spread over all the processors, one instance per processor. In other words, the processor $[i, j]$ will execute the 256 instances $[p\times16+i, q\times16+j]$ with p and q integers between 0 and 255. The equivalent HPF syntax would be:

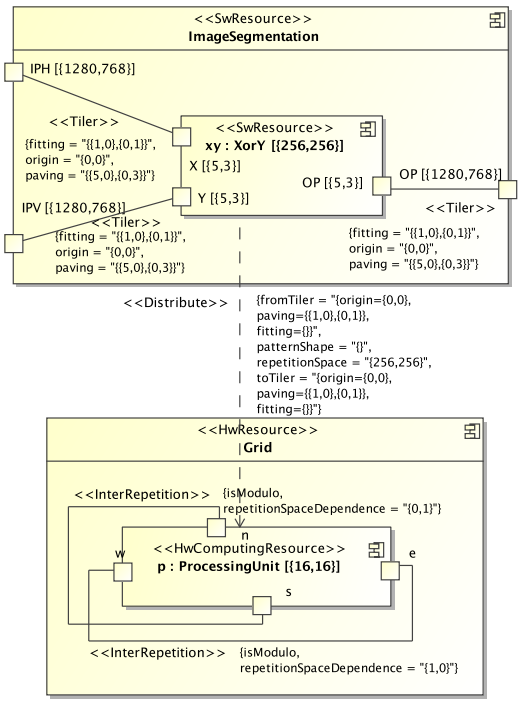

Fig. 9: Cyclic distribution along the two dimensions.

```
!HPF$ DISTRIBUTE xy(CYCLIC,CYCLIC) ONTO P
```

Finally, in Figure 10, the application is distributed in a k-cyclic way along with both dimensions. Blocks of 2×8 application instances are spread over each processor. The processor $[i, j]$ will execute the 256 instances contained in the blocks from $[(p\times16+i)\times2, (q\times16+j)\times8]$ to $[(p\times16+i)\times2+1, (q\times16+j)\times8+7]$ with p between 0 and 127 and q between 0 and 31. The equivalent HPF syntax would be:

```
!HPF$ DISTRIBUTE xy(CYCLIC(8),CYCLIC(2)) ONTO P
```

5) Data distribution: The « Distribute » stereotype also allows distribution of data. In the following, multidimensional data arrays are mapped on instances of memory. On a parallel application, one of the most important aspects affecting performance is the data locality. Knowing that communications between processors penalize the performance of parallel applications, they should be reduced. So data have to be close to the processor which needs it.

Figure 11 illustrates the data distribution of the *IPV* array. We have supposed that the computation is distributed by block, and that each computation works on an area of 5×3 pixels, but also needs as input the neighbor pixels corresponding to this area. So, each *xy* will need 7×5 pixels from *IPV*. This is typical a requirement from a signal processing application. So as to maximize data-locality, the resulting distribution is a mapping of *IPV* as blocks of {82,50} on each memory of the processing units shifted by {80,48}. The borders of each block are duplicated on two processing units. The array is considered

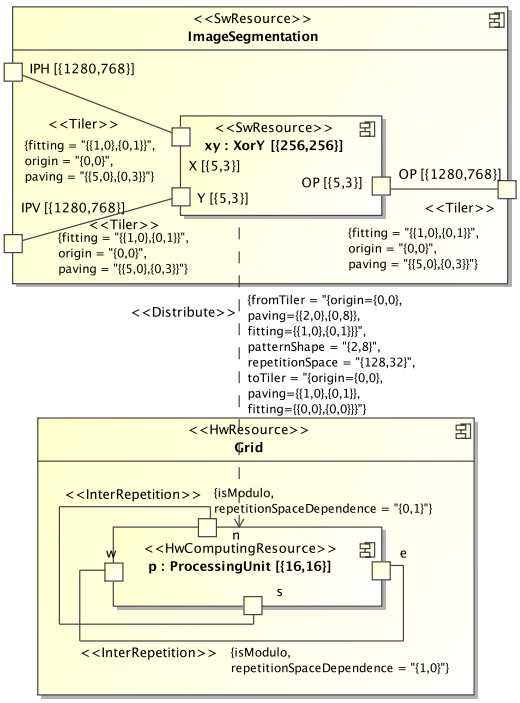

Fig. 10: Task distribution 2-cyclic, 8-cyclic.

toroidal, therefore a border of each first block is shared with the corresponding last block.

V. Conclusion

The MARTE profile provides an allocation mechanism (which is compatible with the mechanism proposed by SysML [13]) to enable the co-design of Systems-on-Chip. The notion of repetition permits to keep a compact design for large regular systems. « Distribute » leverages those two mechanisms to provide the SoC designer the allocation of large applications on parallel architecture with virtually no restrictions on the way the distribution is performed. For example, the freedom of the distribution encompasses all the expressivity available in HPF. It allows the distribution of multidimensional repetitive applicative structures to multidimensional repetitive hardware structures. The fine tuning of the distributions, adapted to the algorithm of the application, permits to get the optimal performances from the system.

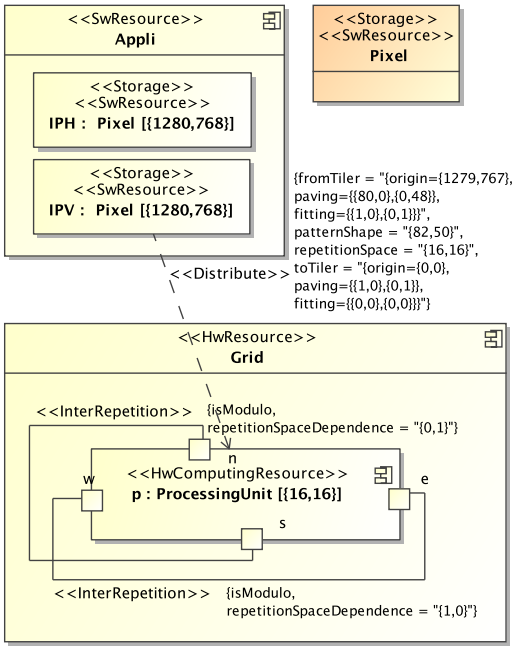

Fig. 11: Data distribution of the array IPV by blocks of {82,50} on each memory of the processing units, with one pixel

References

[1] M. Berekovic and P. Pirsch, "A scalable, distributed network-on-chip architecture for digital signal processing based on simultaneous multithreading (SMT)," in *5th Workshop on Media and Streaming Processor*, San Diego, Dec. 2003.

[2] W. Caarls, P. Jonker, and H. Corporaal, "Skeletons and asynchronous RPC for embedded data and task parallel image processing," *IEICE Transactions on Information and Systems*, vol. E89-D, no. 7, pp. 2036–2043, 2006.

[3] NEC Corporation, "NEC unveils a new class of system LSI solutions, the dynamically reconfigurable processor LSI architecture, at microprocessor forum," http://www.nec.co.jp/press/en/0210/1601.html, San José, California, 2002. [Online]. Available: http://www.nec.co.jp/press/en/0210/1601.html

[4] P. Claydon, "picoChip: A massively parallel array processor," in *Embedded Processor Forum*, San Jose, CA, May 2003.

[5] M. J. Rutten, J. T. van Eijndhoven, E.-J. D. Pol, E. G. Jaspers, P. van der Wolf, O. P. Gangwal, and A. Timmer, "Eclipse: A heterogeneous multiprocessor architecture for flexible media processing," in *IEEE Design and Test of Computers Special issue on Embedded Processor Based Designs*, Ed.: Peter Marwedel, July-August 2002. [Online]. Available: http://home.iae.nl/users/josve/jos/publications/

[6] W. Wolf, "The future of multiprocessor systems-on-chips," in *41st Conference on Design Automation (DAC'04)*, San Diego, California, USA, June 2004.

[7] High Performance Fortran Forum, "High Performance Fortran language specification, version 2.0," Rice University, Houston, TX, Jan. 1997.

[8] Object Management Group, Inc., Ed., *UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP*. http://www.omg.org/cgi-bin/doc?realtime/2005-02-06, Feb. 2005.

[9] ProMarte partners, "A UML profile for marte, initial submission, version 0.44," http://www.omg.org/cgi-bin/doc?realtime/2005-11-01, Nov. 2005.

[10] A. Cuccuru, J.-L. Dekeyser, P. Marquet, and P. Boulet, "Towards UML 2 extensions for compact modeling of regular complex topologies," in *MoDELS/UML 2005, ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems*, Montego Bay, Jamaica, Oct. 2005.

[11] I. R. Quadri, P. Boulet, and J.-L. Dekeyser, "Modeling of topologies of interconnection networks based on multidimensional multiplicity," INRIA, Research Report RR-6201, May 2007. [Online]. Available: https://hal.inria.fr/inria-00149527/en/

[12] P. Boulet, "Array-OL revisited, multidimensional intensive signal processing specification," INRIA, Research Report RR-6113, Feb. 2007. [Online]. Available: http://hal.inria.fr/inria-00128840/en/

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