Design Space Exploration in Application-Specific Hardware Synthesis for Multiple Communicating Nested Loops

Rosilde Corvino, Abdoulaye Gamatié, Marc Geilen, Lech Jozwiak

Design Space Exploration in Application-Specific Hardware Synthesis for Multiple Communicating Nested Loops

Rosilde Corvino*, Abdoulaye Gamatié†,Marc Geilen*, Lech Jóźwiak*
†CNRS/LIFL, Lille, France
Email: abdoulaye.gamatie@lifl.fr
* Technische Universiteit Eindhoven, Eindhoven, The Netherlands
Email: {r.corvino, m.c.w.geilen, l.jozwiak}@tue.nl

***Abstract*—Application specific MPSoCs are often used to implement high-performance data-intensive applications. MPSoC design requires a rapid and efficient exploration of the hardware architecture possibilities to adequately orchestrate the data distribution and architecture of parallel MPSoC computing resources. Behavioral specifications of data-intensive applications are usually given in the form of a loop-based sequential code, which requires parallelization and task scheduling for an efficient MPSoC implementation. Existing approaches in application specific hardware synthesis, use loop transformations to efficiently parallelize single nested loops and use Synchronous Data Flows to statically schedule and balance the data production and consumption of multiple communicating loops. This creates a separation between data and task parallelism analyses, which can reduce the possibilities for throughput optimization in high-performance data-intensive applications. This paper proposes a method for a concurrent exploration of data and task parallelism when using loop transformations to optimize data transfer and storage mechanisms for both single and multiple communicating nested loops. This method provides orchestrated application specific decisions on communication architecture, memory hierarchy and computing resource parallelism. It is computationally efficient and produces high-performance architectures.**

I. Introduction

Data intensive applications require high computing performance and parallelism as provided by application specific Multi-Processor Systems-on-Chip (MPSoCs). But in MPSoC design, application parallelism can be adequately exploited only if internal mechanisms orchestrate an optimized and bottleneck-free distribution of data to the different parallel computing resources. Hence, the extremely complex design of such MPSoCs should benefit from abstract analysis and synthesis methods, which focus on data-oriented design aspects, such as the parallelism level of computing resources and the data transfer and storage micro-architecture [1], i.e. communication structure and the memory hierarchy.

The design of MPSoCs for data-intensive applications has been studied by many researchers (cf. Section II) and it often focuses on two abstraction levels: 1) system level, where an abstract analysis targets the synthesis of communication and storage mechanisms; 2) processor level, where techniques such as High Level Synthesis (HLS) may solve problems as application instructions parallelization, scheduling and mapping and generate Register Transfer Level descriptions from a high level sequential specification.

Loop-based analysis and synthesis methods have been largely used, at the processor level, to explore different design possibilities, e.g., estimate the storage requirements [2], improve the instruction level parallelism (ILP) [3], optimize the loop iteration scheduling, reduce the redundant memory traffic and improve the synthesis of computing data paths. Nevertheless, most of existing works using loop transformations for hardware synthesis fail to capture the interaction between different loop nests [4]. These methods usually exploit hardware characteristics related to execution of a single loop nest, such as the instruction level parallelism or the memory reuse. In contrast, more abstract methods based on synchronous data flows (SDFs) [5] are commonly used to explore system communication structure and memory hierarchy, allowing for data parallelism exploration and exploitation in systems with *multiple communicating loop nests*. Unfortunately most SDF models do not take into account the multidimensionality of the transferred data, and, consequently, they are not well-suited to describe the effects of loop-transformation-like operations that can be used to efficiently explore the data parallelism of an application.

Our contribution consists in a design space exploration (DSE) method and tool for the design of data intensive computing systems executing communicating loop nests. More specifically, our contribution includes:

1) A tool for an automatic DSE of data-intensive systems with multiple communicating loop nests.
2) An abstract customizable model of the MPSoC architecture.
3) A formal method, based on abstract clocks [6], to capture the application schedule and mapping, as well as, quality indicators of the MPSoC design.

Our method and tool exploit an application model, called Array-OL [7], which can be analyzed as a multidimensional SDF model [8] for static scheduling, but Array-OL is more suitable than SDF to efficiently describe *data-oriented loop*

transformations [7]. Our method explores different loop transformations for an application with *multiple communicating nested loops*. From these transformations, it infers some promising *architecture template customizations*. In order to make our analysis efficient, *abstract clocks* [9] are used to capture the scheduling and mapping information of the iterative tasks when mapped onto our customized architecture template. The abstract clocks also capture the scheduling and mapping modifications due to the loop transformations. They are also used to infer values of some *quality indicators* such as internal memory size, energy consumption and system throughput, assessing the exploration results. The exploration process is performed through a genetic algorithm, where the genes encode the applied loop transformations; the explored solutions are represented by abstract clocks and the quality indicators provide information for the best solution selection.

The rest of the paper is organized as follows: Section II presents the related works; Section III introduces the specification formalisms and the architecture template used in our approach; Section IV provides an overview of our exploration method and details its implementation; Section V discusses the exploration results for four applications case studies. Finally, Section VI presents the conclusion and future research directions.

II. Related Works

Genetic algorithms [10], [11] demonstrated their effectiveness in the multi-objective exploration of large design spaces. Most of the presented methods construct the hardware architectures from a set of possible pre-existing components. In this paper, we use a data-oriented customizable architecture template with customization parameters, values of which are directly inferred from the application structures. As a result, the space of the analyzed hardware solutions is narrowed to the solutions that are appropriate for the considered application with a consequent increase of the exploration efficiency.

The exploitation of loop transformations for the system architecture design and synthesis was already proposed in the 1990s [12], [13], and further enhanced by recent works such as [14], [3] and [2]. Most of these works optimize single loop nests for mainly two kinds of objectives: the instruction level parallelism optimization [12], [14] and the memory subsystem optimization in terms of memory re-use [13], [2] or redundant memory accesses reduction [3].

In contrast with these existing methods, our method is aimed at the orchestrated exploration of the communication and memory system architecture and the computation resource parallelism. It selects application-specific optimized solutions allowing for massive data parallelism and, as a consequence, targets multiple communicating loop nests, i.e. loop nests that exchange multidimensional data. Our proposition can be used complementary to the previous solutions in a more general framework: our method optimizes the data transfer and storage architecture at the system level, while the previous methods improve the ILP and reduce redundant memory accesses inside single loop bodies.

Methods capturing interactions between loop nests are quite unique, involve some serious limitations and are not automated. For instance, [4] has the following limitations with respect to our proposal: it is not automated, it does not consider the latency of the communication with an external memory or bus, it does not support scratch-pad memories, and it does not consider the data granularity as a parameter of the design exploration. Due to the high problem complexity, our method has still several limitations, e.g. it only supports applications with a predictable behavior. However, it has the advantages of being fully automated, rapid and efficient. In [15] and [16], we introduced and formalized the problem of DSE of data-intensive systems with multiple communicating loop nests, through the usage of integer constraints. In this paper, we present a holistic DSE framework and enrich the problem formalization by replacing the value of integer variables with binary words that encode in addition the order of data accesses, the time needed to read data and synchronizations. A more detailed comparison of the two formalisms is given in [17].

III. Application and architecture specification

In this section, we introduce the Array-OL formalism, the customizable architecture template and the abstract clocks formalism, on which our approach is built.

A. Application specification

An Array-OL application specification [7] involves a set of tasks that can be elementary, compound or repeated. For our method, the repeated tasks are of main interest. They consume and produce multidimensional data arrays piecewise, where each piece is referred to as a pattern. The Array-OL application specification gives information from which one can derive the data parallelism, i.e. data dependencies, data access patterns and repetition space of the tasks. Fig. 1 illustrates an Array-OL specification for a JPEG encoder.

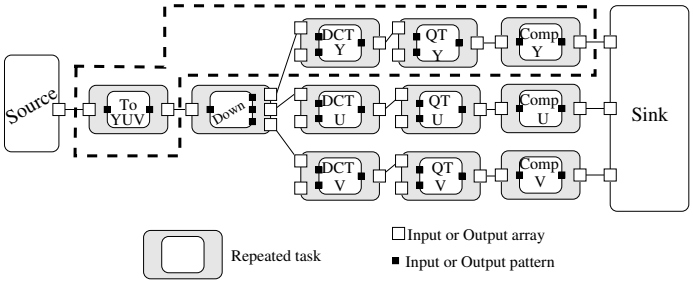

Fig. 1. Array-OL Specification of a JPEG encoder. A gray-scale JPEG encoder only contains the tasks enclosed by the dotted lines.

Fig. 1 shows that, at a level describing interactions between the tasks, Array-OL captures task dependencies and data granularity of the tokens consumed and produced by each task. At this inter task level, Array-OL model is equivalent to a SDF model.

Fig. 2 shows that, at a level capturing the behavior of a single repeated task, being equivalent to a loop nest, Array-OL exploits the polyhedral model of the task, specifying the

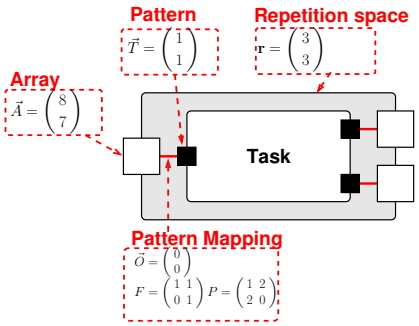

Fig. 2. Array-OL task specification and tilers.

repetition domain of the loop (r), the size of the consumed and produced arrays ($\vec{A}$), the data dependencies given by the mapping between input and output patterns ($\vec{T}$) and specifying, with two additional matrices P and F, how the patterns are positioned in the array (with P) and how the data are positioned in a pattern (with F). The positions are computed with respect to the origin of the data array.

Loop-transformation-like operations can be applied to an Array-OL specification, such as: 1) task fusion, which merges iterative tasks in the same iteration space; 2) tiling, which adds a level of depth to a loop nest. (In our method, the tiled iterations are systematically flattened and this is equivalent to unrolling of iterations in sequentially executed loops.) 3) paving-change, which changes the data granularity of a repeated task. This last transformation corresponds to a vectorization in a sequential code.

B. Hardware architecture template

Our hardware architecture template, shown in FIG. 3, represents a simplified model of a tile-based MPSoC [18], [19]. Each processing tile (Proc Tile) contains processing elements, local memories (LM) and a local control for data access (CTRL). Thanks to a double buffering mechanism (i.e., two LMs alternatively read and written by the processing elements and the CTRL of a processing tile), the data accesses and computations can be performed in parallel.

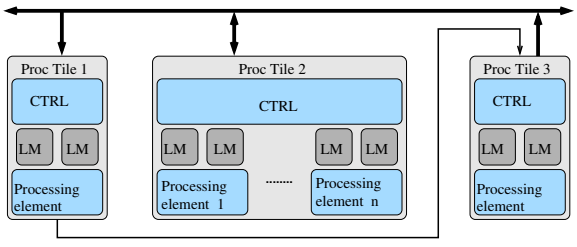

Fig. 3. An architecture template with three processing tiles.

For this architecture template, a VHDL library of customizable elements has been developed, including local memories, controllers, processing elements, processing tiles and busses. The elements of the library can be composed together to design an application specific hardware, and are customizable with respect to the number and size of local memories, the number of processing elements in a processing tile and the type of communication between the processing tiles.

For instance, processing tiles frequently exchanging small amounts of data can communicate through point-to-point links, while processing tiles exchanging large amounts of data can communicate through a shared memory, which is accessed through a shared bus. The MPSoC architecture is customized according to the results of a DSE, which evaluates and selects transformations of the Array-OL specification effectively exploiting the application parallelism and efficiently using the hardware resources.

Mapping and scheduling rules assign each Array-OL repeated task to a processing tile of the MPSoC and each tile of data produced and consumed by a task to a local double buffer. The loop transformations of the application mentioned in Section III-A directly set some mapping and scheduling rules, as follows: 1) The task fusion determines the communication structure. Indeed, when two tasks are merged they repeatedly exchange small multidimensional data blocks. Therefore, they are mapped onto a pipeline of processing tiles with point-to-point communication between them. They also benefit from parallel read and write accesses to the local double buffers. By contrast, two unmerged tasks exchange large multidimensional arrays, that cannot be stored internally. Consequently, they are mapped onto processing tiles communicating via the shared bus with exclusive read and write memory access modes. 2) The paving-change is used to explore different sizes of local double buffers. 3) The tiling (which in our case is systematically flattened and unrolled) is used to increase of the parallelism level of the computing resources by multiplying the number of processing elements in a processing tile.

After mapping an application onto a corresponding instance of the generic tile-based MPSoC template, the execution of the application is described by abstract clocks.

C. Abstract clocks

Abstract clocks describe how the data consumption and production of the repeated Array-OL tasks are synchronized when they are executed on MPSoC processing tiles according to the previously explained mapping rules. Consequently, one abstract clock is associated with each input and output data tile of an Array-OL model. *An abstract clock is a periodic binary word, which has a phase* Φ *and a period* Π *repeated* r *times*. Within a period, 0 denotes a synchronization cycle and 1 denotes a read or write data access. For instance, FIG. 4 gives the abstract clocks describing the behavior of a repeated task when executed onto a processing tile. The considered task has

$$
\begin{aligned}
clk(i0) &= \overbrace{0000001\ 0000001\ 0000001\ 0000001}^{r}\ \ldots \\
clk(i1) &= 0001111\ 0001111\ .\ .\ \ldots \\
clk(o1) &= \underbrace{0000000}_{\Phi}\ \overbrace{\underbrace{0000011}_{\Pi}\ \underbrace{0000011}_{\Pi}}^{r}\ .\ \ldots
\end{aligned}
$$

Fig. 4. Abstract clocks associated with a repeated task having two inputs $i0$ and $i1$, and one output $o1$.

two inputs $i0$ and $i1$, and one output $o1$. The tile cardinality, i.e. the number of data in each task tile, is respectively 1, 4 and 2 through $i0$, $i1$, and $o1$. The clock associated with the port $o1$, named $clk(o1)$, has a phase Φ, marked by 0's, to synchronize the first output firing to the input availability. All the clocks of a task have the same period Π during which the rhythm of data consumption and production is marked by 1's. The clock cycles marked with 0's within a clock period or a clock phase can represent the latency due to an external memory accesses.

The loop transformations of Section III-A affect the data consumption and production rates of a task and, as a consequence, the associated abstract clocks. For example, the tiling modifies the repetition space of a task and, as a consequence, the parameter r of the associated abstract clock; the paving-change modifies the number of read or written data and, in result, the number of 1's in a clock period. These modifications can be represented by integer multiplication factors called *transformation factors* that can be used to efficiently encode a set of transformations.

Moreover, from the abstract clocks we can compute three **quality indicators** of the application restructuring and related MPSoC platform customization to assess the obtained solutions and guide the design exploration.

1) Amount of internal memory:

$$IM = \sum_t \left\{ \sum_i \{2 \times |\Pi_t(i)|_1\} \right\} + \sum_{t \to sink} \left\{ \sum_o \{|\Pi_t(o)|_1\} \right\}$$

where $|\Pi_t(i)|_1$ (respectively $|\Pi_t(o)|_1$) indicates the number of 1's in the period Π_t of an input port i (respectively output port o) of a task t. The factor $|\Pi_t(i)|_1$ represents the number of input data needed to be available on the port i so that the task t can fire $|\Pi_t(o)|_1$ outputs. The factor 2 is due to the double buffering mechanism. The index $t \to sink$ indicates all tasks t communicating with a sink.

2) Throughput:

$$T = \frac{\sum_{t \to sink} \{\sum_o \{|\Pi_t(o)|_1\}\}}{\sum_p \left\{ r \times \Delta_{\Pi_p} \right\}}$$

where the numerator is the total number of produced output data and the denominator is the latency of the computations needed to produce the total output data.

In the numerator, o indicates an output port of a task t communicating with a sink. In the denominator, Δ_{Π_p} is a latency computed as $\Pi_p \times r$, where Π_p is the period of tasks merged in a pipeline p and r is the number of times this period is repeated until the output data of the pipeline are produced.

3) Energy consumption due to the external memory accesses.

$$E = E_{read} \times \sum_{t \leftarrow source} \sum_i |\Pi_t(i)|_1 + E_{write} \times \sum_{t \to sink} \sum_o |\Pi_t(o)|_1$$

where $|\Pi_t(.)|_1$ is defined as in above formulas. E_{read} and E_{write} are the energy consumption per read and write depending on the used technology and the size of the external memory. In our explorations, we use 0.35 μm SRAM. The size EM of the external memory is the sum of all the input and output array sizes, for all the tasks communicating with the source and sink tasks. It depends on the transformations applied to the Array-OL specification. Given EM, $E_{read} = a_r + b_r \times EM$ and $E_{write} = a_w + b_w \times EM$, with $a_r = 6.37504 \times 10^{-4}$, $b_r = 1.186 \times 10^{-1}$, $a_w = 4.75004 \times 10^{-4}$ and $b_w = 3.65 \times 10^{-2}$.

IV. Design Exploration Method Overview

The proposed design method consists in: 1) exploring different transformations of an application, 2) from this, inferring a set of parameters that customize the generic architecture template and, 3) finally, among the possible solutions, selecting the application transformations and corresponding architecture parameters that optimize the performance and efficiency of the architecture design.

The inputs of the method are an Array-OL behavioral specification of an application and a customizable hardware architecture template. The Array-OL specification is iteratively restructured when exploring the loop transformations explained in Section III-A.

A direct mapping and coarse data-oriented scheduling causes that each of these application transformations corresponds to an architecture customization, as specified by the mapping and scheduling rules of Section III-B. These mapping and scheduling results are described by abstract clocks.

Quality indicators, defined in Section III-C and calculated from the abstract clocks, are used to evaluate the design exploration outputs and guide the exploration itself towards the optimal solutions.

The outputs of the design exploration are Pareto pairs consisting of a restructured application and an optimized set of hardware architecture parameters, to which abstract clocks are associated.

A. Exploration steps

A DSE tool implementing the proposed method is shown in Fig. 5. It has been implemented using Java and involves three exploration steps: 1) an exhaustive enumeration of all possible fusions; 2) for each possible fusion, a heuristic exploration of the other loop transformations: loop tiling and paving-change. (This exploration is performed with Opt4J [20] a modular framework supporting genetic algorithms;) 3) a final selection of Pareto solutions. Following is a description of these three steps.

1) Fusion enumeration divides the space of all possible task fusions in sub-spaces that can be independently explored. As in [21], [16], the number of sub-spaces is equal to the number of integer partitions of n, where n is the number of tasks in the application specification. An integer partition is a set of positive integer vectors whose components add up to n. For example, the integer partition of $n = 4$ is $[1, 1, 1, 1]$, $[2, 1, 1]$, $[2, 2]$, $[3, 1]$, $[4]$. For this example, the integer partition $[4]$ maps all the possible fusions between 4 tasks of the application. This mapping extremely reduces the number of possible fusion comparisons, because only fusions belonging to the same sub-space are compared with each other. However, due to the exhaustive fusion enumeration and the successive

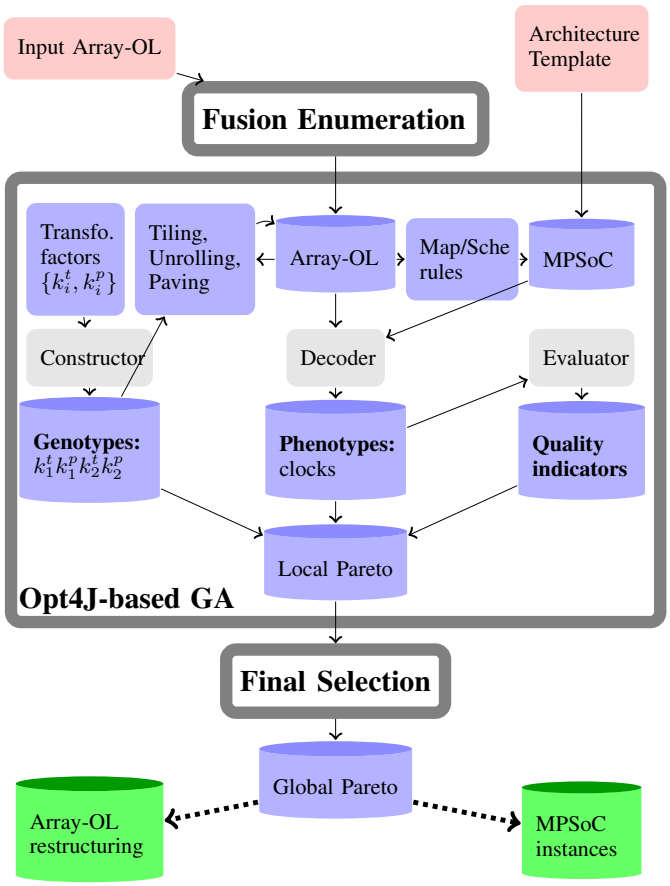

Fig. 5. Flow of the proposed method.

genetic algorithm, a good quality of MPSoC customization results is ensured.

2) Opt4J-based genetic algorithm (GA). For each subspace mapping a n integer partition, the Opt4J-based GA explores the tiling and paving change and finds the *local Pareto solutions*. This Opt4J-based GA implementation is discussed in the next section. All the *local Pareto solutions* are merged in a new exploration space and passed to the final selection.

3) Final selection. This algorithm is an exhaustive search of the new formed exploration space in order to find *the global Pareto front*. It exhaustively explores spaces of reasonable sizes because it only considers a union of *local Pareto solutions* selected by Opt4J-based GA.

We decided to use an exhaustive exploration method for the fusion and a heuristic method for the other transformations, because there is a finite and reasonably low number of possible fusions, while the number of possible tiling and paving-change is considerably high.

B. Opt4J and the genetic algorithm

As all genetic algorithms, Opt4J [20] evolves a population of individuals with respect to optimization objectives. Promising individuals are selected as parents and are combined through a cross-over operation to generate offsprings. The new generated offsprings undergo a genetic mutation that avoids unwanted convergence to local optima and maximizes the probability to find global optimal solutions of the design problem. The individual characteristics are encoded by genes and are manifested by phenotypes.

In Opt4J-based GA exploration (cf. FIG. 5), a **constructor** generates the genes of the individual in a initial population. A **decoder** computes the phenotype of the individuals from their genes. An **evaluator** assesses the quality of the individuals with respect to optimization criteria. The usage of Opt4J requires an adequate definition of genotypes, phenotypes and objectives criteria. In our method:

The genotypes are strings of transformation factors (cf. FIG. 5 and Section III-C), e.g. for an application with 2 tasks $gene = String\{k_1^t, k_1^p, k_2^t, k_2^p\}$, where k_i^t is the tiling factor of task i and k_i^p is the paving-change factor of task i. These genes are used by the decoder to infer the transformations of the Array-OL specification. The *direct mapping and coarse scheduling* infer the hardware template customization from each analyzed restructuring of an Array-OL specification. The decoder produces phenotypes as output, which concisely capture relevant information on a hardware template customization.

The phenotypes are the abstract clocks capturing the mapping and scheduling rules associated to a restructured Array-OL specification and the corresponding architecture template customization.

The objectives criteria are represented by the quality indicators, as defined in Section III-C. They are obtained from abstract clocks and are associated with each individual, i.e. a design output.

V. EXPERIMENTAL RESULTS

To demonstrate the validity of our method, we have analyzed four applications that are briefly described in Table I.

Application name	number of tasks	description
JPEG enc	11	Described in FIG. 1
STAP	7	Radar application [7]
VBL	6	Hydrophone monitoring [22]
LPSF	4	Low pass spatial filter

TABLE I
APPLICATION DESCRIPTIONS.

In the rest of this section, we first present the experiment conditions (Section V-A), and discuss the experimental results. These results are classified in two main categories: results demonstrating the advantages of the implemented automatic method (Section V-B) and results showing the quality of the selected hardware solutions (Section V-C).

A. Experiment conditions and Opt4J GA settings

The genotypes used by Opt4J are computed from a set of possible transformation factors for the tiling (k_i^t) and for the paving-change (k_i^p). These factors are selected among the following possible values: $k_i^t = k_i^p = \{1, 2, 4, 8\}$. We choose values among powers of 2 to reduce the complexity of

hardware (HW) implementation. We put the maximum value to 8 to cap the area overhead due to the internal memories and computing resources parallelism. The Opt4J genetic algorithm has the settings described by Table II.

Opt4J parameter	set values
population size	100 individuals
generation number	100 individuals
number of parents elected for reproduction (i.e. crossover)	25 individuals
number of children elected to integrate the new generation	25 individuals
crossover rate	95 %

TABLE II
OPT4J PARAMETER SETTING

The Opt4J parameter setting has been chosen from a set of trials demonstrating that the accuracy of the exploration was not improved by a further increase of parameter values.

The analyzed explorations were performed on an Intel i7 quad core processor running at 2.67 GHz with 4G of RAM. As we used a random generated initial population in Opt4J, and computed the average of exploration results over 10 trials all with the settings of Table II.

B. Results characterizing the exploration method

Table III gives the complexity of the performed exploration, for each analyzed application. For example, a JPEG encoder has 11 tasks and 75 possible fusions. For each fusion, the initial population of the associated genetic algorithm includes 100 individuals, it evolves for 100 generations and at each evolution 25 new individuals are created and explored. Thus in total, 10000 individuals are explored. This is a reasonable number with respect to the whole explorable space containing $4^n \times 75 \approx 315M$ individuals. The final selection provides, on average, 11 Pareto solutions. The number of selected

	JPEG	STAP	VBL	LPSF
tasks in the application	11	7	6	4
num. of possible fusions	75	54	54	8
explored individuals	10000	7900	7900	3200
whole exploration space (individuals)	$\approx 315M$	$\approx 900K$	$\approx 900K$	$\approx 2K$
average number of global Pareto solutions	11	7	2	1

TABLE III
EXPLORATION COMPLEXITY AND SELECTIVITY.

global Pareto solutions (P) increases with the problem size, i.e. the number of input tasks of the application. Indeed the number of times that the population evolves, is the same for different applications and thus, the complex applications are proportionally less exhaustively optimized than the simpler ones. One could of course adapt the customization parameters of Opt4J GA in table II to the application complexity.

Table IV gives the run-time of the whole exploration for all the applications; it also gives the percentage of the run-time for each exploration step: fusion exploration, genetic inside Opt4J and final selection + others, where "others" refers to the time spent to read and write text files. First of all, we observe that, as expected, the run-time of the exploration grows with the size of the exploration space. Then, for all the analyzed examples, the most of the run-time is spent to perform the genetic algorithm in Opt4J. The percentage of the time spent to perform the other exploration steps is negligible. The percentage of time spent to perform the different steps is almost invariable with respect to the different applications and different trials per application. Furthermore, the total run-time (see Table IV) is of order of seconds even for a significant complexity of a solved problem (see Table III).

	JPEG	STAP	VBL	LPSF
run-time of exploration (sec.)	81	37	34	5
% of run-time for Fusion Exploration	3%	3%	3%	2%
% of run-time for Opt4J	96%	97%	97%	98%
% of run-time for Final Selection + others	1%	≈0%	≈0%	≈0%

TABLE IV
EXPLORATION RUN-TIME.

To evaluate the precision of our exploration, we compared our method to an exhaustive search exploring all the application transformations inferred from all the possible combinations of transformation factors. Table V summarizes this comparison. It gives the run-time of the two exploration methods and the ϵ-indicators [23], i.e. indicators asserting the quality of a Pareto front with respect to another. In particular $\epsilon = 1$ if the two Pareto fronts are of identical quality.

	run-time		ϵ-indicator
	our method	exhaustive	
LPSF	5 sec.	14 min.	1
VBL	34 sec.	n.a.	-
STAP	37 sec.	n.a.	-
JPEG	81 sec.	n.a.	-
YJPEG	7 sec.	16 min.	1
STAP*	35 sec.	21 min.	1
VBL*	33 sec.	19 min.	1

TABLE V
QUALITY OF PARETO FRONT SEARCH.

Table V shows that it was not possible to apply the exhaustive method to JPEG, STAP and VBL applications. Indeed, the heap memory (≈4GB) was exceeded. In consequence, we performed the comparison for a gray-scale JPEG encoder (YJPEG in Table V), and two reduced explorations of STAP and VBL applications with only two possible tiling and paving factors $k_i^t = k_i^p = \{1, 2\}$. These two last explorations are denoted STAP* and VBL* in Table V. For all the explorations that were possible to compare, the ϵ-indicators are 1.

All these observations demonstrate that the proposed method provides an effective and rapid exploration of a large design space.

C. Example of a complete exploration: JPEG encoder

We have assessed the quality of the hardware architectures selected by our DSE method, by comparing the performances of our selected hardware architectures for the JPEG encoder modeled in FIG. 1, against solutions existing in the literature. We analyze the results of two Virtex-4 FPGA implementations, one for a gray-scale JPEG and one for a color JPEG encoder.

Our FPGA implementations emulate only the data transfer and storage mechanisms. In result, the area of our implementations involves an overhead comparing to the published solutions. Furthermore, the published solutions are implemented on older FPGA platforms (a Flex 10 KE FPGA of Altera nad a Virtex-II FPGA of Xilinx) with different performance and architectural characteristics. In consequence, these results have to be considered an indication of the possible improvements that can be obtained with our method.

We compared our implementation of the color JPEG encoder with the implementation presented in [24], which is obtained with the HLS tool ImpulseC, targets a Virtex-II and is 100 times faster than a DSP-based implementation. The authors in [24] are able to process 41,000 blocks of 8x8 per second with a clock frequency of 50 MHz. While we achieve a throughput of 312,500 blocks of 8x8 pixels per second for the highest throughput Pareto solution and can synthesize this solution with a maximum clock frequency up to 150 MHz. In our implementation, the local memories are implemented into dedicated FPGA RAMs. Thus, their area occupancy is optimized. Our design implementation occupies 6% of slices and 88% of RAMB16s. The number of used slices for logic is quite low and leaves room for implementation of the processing elements. We have compared our implementation of a gray-scale JPEG encoder with the manual implementation proposed in [25]. We achieve a throughput of 164 frames of 640×480 pixels compared to a throughput of 122 frames achieved in [25]. For an area occupancy of 1% of slices and 8% of RAMB16s.

A more accurately described case study of a color JPEG automatically selected architecture, which is implemented on a Virtex-6 FPGA and which outperforms the OpenCores JPEG encoder presented in [26], is detailed in [27].

These experiments show that our implementations results in a quite low logic area overhead, and efficiently exploit the FPGA RAMB16s to achieve a significant throughput increase.

VI. Conclusion

In this paper, we proposed a new method for the design of MPSoCs for data-intensive applications with multiple communicating nested loops. Such a method uses loop transformations to perform a combined optimization of the hardware communication structure, the memory hierarchy and the computing resource parallelism. Experimental results showed that the proposed method provides an effective and rapid exploration of large design spaces. The analysis of the selected solution implementations demonstrates that the implementations achieve a significant throughput increase with a low area overhead due to the logic and an efficient exploitation of the FPGA integrated RAMs. To the best of our knowledge, this method is the first one using loop transformations to explore in combination the communication, memory and computing architectures. Several future research directions are possible, such as integrating into our method additional relevant loop transformations; enlarging the list of possible mapping and scheduling rules, providing the possibility to decide based on the application characteristics which loop transformations should be used and guide the DSE; applying our method to the case of Application Specific Instruction Set Processors.

VII. Acknowledgement

This paper has been partially supported by ASAM project of the ARTEMIS Research Program.

References

[1] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg, "Data and memory optimization techniques for embedded sys." *ACM Trans. on Design Automation of Electronic Sys.*, vol. 6, pp. 149–206, April 2001.

[2] Q. Hu, P. G. Kjeldsberg, A. Vandecappelle, M. Palkovic, and F. Catthoor, "Incremental hierarchical memory size estimation for steering of loop transformations," *ACM Trans. on Design Automation of Electronic Sys.*, vol. 12, no. 50, Sept. 2007.

[3] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau, "Spark : A high-level synthesis framework for applying parallelizing compiler transformations," in *Proc. of the 16th International Conference on VLSI Design, 2003*.

[4] Y. Bouchebaba, B. Girodias, G. Nicolescu, E. Aboulhamid, B. Lavigueur, and P. Paulin, "Mpsoc memory optimization using program transformation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, September 2007.

[5] C. Lee, S. Kim, and S. Ha, "A systematic design space exploration of mpsoc based on synchronous data flow specification," *J. of Signal Processing Systems*, vol. 58, pp. 193–213, March 2010.

[6] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[7] C. Glitia, P. Boulet, E. Lenormand, and M. Barreteau, "Repetitive model refactoring strategy for the design space exploration of intensive signal processing applications," *J. of Systems Architecture*, vol. 57, pp. 815 – 829, Oct. 2011.

[8] P. Murthy and E. A. Lee, "Multidimensional synchronous dataflow," *Signal Processing, IEEE*, vol. 50, pp. 2064–2079, Aug. 2002.

[9] A. Cohen, M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau, and M. Pouzet, "N-synchronous kahn networks: a relaxed model of synchrony for real-time systems," in *Proc. of the ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '06)*.

[10] T. Basten, E. van Benthum, M. Geilen, M. Hendriks, F. Houben, G. Igna, F. Reckers, S. de Smet, L. Somers, E. Teeselink, N. Trka, F. Vaandrager, J. Verriet, M. Voorhoeve, and Y. Yang, "Model-driven design-space exploration for embedded systems: The octopus toolset," in *Leveraging Applications of Formal Methods, Verification, and Validation*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds. Springer Berlin / Heidelberg, 2010, vol. 6415, pp. 90–105.

[11] M. Lukasiewycz, M. Streubühr, M. Glaß, C. Haubelt, and J. Teich, "Combined system synthesis and communication architecture exploration for mpsocs," in *Proc. of the Conference on Design, Automation and Test in Europe (DATE'09)*.

[12] T.-F. Lee, A. C.-H. Wu, Y.-L. Lin, and D. D. Gajski, "A transformation-based method for loop folding." *IEEE Trans. on CAD of Integrated Circuits and Systems*, pp. 439–450, 1994.

[13] D. Kolson, A. Nicolau, and N. Dutt, "Elimination of redundant memory traffic in high-level synthesis," *IEEE Trans. on Comp-aided Design*, vol. 15, pp. 1354–1363, 1996.

[14] J. Park, P. C. Diniz, and K. R. Shesha Shayee, "Performance and area modeling of complete FPGA designs in the presence of loop transformations," *IEEE Trans. Comput.*, vol. 53, pp. 1420–1435, November 2004.

[15] R. Corvino, A. Gamatie, and P. Boulet, "Design space exploration for efficient data intensive computing on socs," in *Handbook of Data Intensive Computing*, B. Furht and A. Escalante, Eds. Springer New York, 2011, pp. 581–616.

[16] R. Corvino, A. Gamatié, and P. Boulet, "Architecture exploration for efficient data transfer and storage in data-parallel applications," in *Euro-Par 2010 - Parallel Processing*, ser. Lecture Notes in Computer Science, P. DAmbra, M. Guarracino, and D. Talia, Eds. Springer Berlin / Heidelberg, 2010, vol. 6271, pp. 101–116.

[17] R. Corvino and A. Gamatie, "Abstract clocks for the dse of data-intensive applications on mpsocs," in *ISPA 2012 4th IEEE International Workshop on Multicore and Multithreaded Architectures and Algorithms*, 2012.

[18] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, pp. 70 –78, jan 2002.

[19] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of Design Automation Conference (DAC'01)*, 2001.

[20] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J - A Modular Framework for Meta-heuristic Optimization," in *Proc. of the Genetic and Evolutionary Computing Conference (GECCO 2011)*.

[21] T. Rahwan, S. Ramchurn, N. Jennings, and A. Giovannucci, "An anytime algorithm for optimal coalition structure generation," *J. of Artificial Intelligence Research (JAIR)*, vol. 34, pp. 521–567, April 2009.

[22] P. Boulet, J.-L. Dekeyser, J.-L. Levaire, P. Marquet, J. Soula, and A. Demeure, "Visual data-parallel programming for signal processing applications," in *Proc. of Ninth Euromicro Workshop on Parallel and Distributed Processing, 2001*.

[23] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Trans. on Evolutionary Computation*, vol. 7, pp. 117–132, April 2003.

[24] S. Thibault. (2011, Sept.) FPGA for DSP: A JPEG encoder case study. [Online]. Available: http://www.gmvhdl.com/fpga_for_dsp.html

[25] L. V. Agostini, R. C. Porto, S. Bampi, and I. S. Silva, "A FPGA based design of a multiplierless and fully pipelined JPEG compressor," in *Proc. of the 8th Euromicro Conference on Digital System Design (DSD'05)*.

[26] JPEG Encoder :: Overview. [Online]. Available: http://opencores.org/project,mkjpeg

[27] R. Corvino, E. Diken, A. Gamatie, and L. Jozwiak, "Transformation-based exploration of data parallel architecture for customizable hardware: A jpeg encoder case study," in *DSD 2012*, 2012.