Transformation-based Exploration of Data Parallel Architecture for Customizable Hardware: A JPEG Encoder Case Study

Rosilde Corvino*, Erkan Diken*, Abdoulaye Gamatié†, Lech Jóźwiak*
* Technische Universiteit Eindhoven, Eindhoven, The Netherlands
Email: {r.corvino, e.diken, l.jozwiak}@tue.nl
†CNRS/LIFL, Lille, France
Email: abdoulaye.gamatie@lifl.fr

***Abstract*—In this paper, we present a method for the design of MPSoCs for complex data-intensive applications. This method aims at a blend exploration of the communication, the memory system architecture and the computation resource parallelism. The proposed method is exemplified on a JPEG Encoder case study by describing all the design steps. Our method allows for a JPEG encoder implementation having a throughput increase of 84% and an increase of the achievable FPGA maximum frequency f_{max} of 64% with an area overhead of 6× with respect to a reference solution. Our method is also assessed with additional explorations of applications from different domains.**

***Index Terms*—Design space exploration, loop transformations, data parallelism, design automation, data transfer and storage architecture**

I. Introduction

Multiprocessor System on Chip (MPSoC) are capable of serving the requirements of new data-intensive streaming applications with an adequate high computing performance and parallelism. However, the high complexity of MPSoC design represents an obstacle in exploiting MPSoC capabilities and may eventually prevent from meeting the design constraints of streaming applications. The MPSoC design complexity has a twofold nature. On one hand, it concerns the deployment of an *ad-hoc* number of parallel computing resources that should correctly serve the application requirements. On the other hand, it concerns the design of a data transfer and storage micro-architecture [1] that should guarantee a bottleneck-free highly optimized data distribution to the MPSoC parallel computing resources.

The design of MPSoCs for streaming applications has been a hot research topic for the last two decades and, in general, its complexity is addressed at two abstraction levels: 1) the system level [2], [3], [4], targeting the synthesis of system communication and storage mechanisms; 2) the processor level [5], targeting the computing kernel parallelization, scheduling and mapping. However, such a separation of concerns between communication structure and computing resource design, which is required to handle the complexity of MPSoCs design, introduces a separation between data-related design concerns that, especially in the case of data-intensive streaming applications, should be addressed and solved within a unique optimally orchestrated solution.

Processor design for streaming applications has benefit from loop-based High Level Synthesis (HLS) [5], which takes as input a loop-based C-code specifying the streaming behavior of an application and produces as output a Register Transfer Level (RTL) model, usually realizing a VLIW machine. Such loop-based HLS methods largely exploit loop transformations, such as loop unrolling, skewing and tiling, to enhance data locality and application parallelization possibilities [6], [7], [8], [9], [10]. The ultimate aim of these loop transformations is to optimize the result of the processor architecture design and synthesis by improving the quality of the input application specification. In particular, these techniques are used to estimate the storage requirements [10], improve the instruction level parallelism (ILP) [9], [6], [8], optimize the loop iteration scheduling, reduce the redundant memory traffic [9] and improve the memory re-use [7], [10].

However, most of existing works using loop transformations for hardware synthesis fail to capture the interaction between different loop nests [11] and optimize the synthesis of computing data paths only for single loop nests. In contrast, more abstract methods based on synchronous data flows (SDFs) [12] are commonly used to model system communication structure and memory hierarchy, allowing for static scheduling and mapping analysis of streaming applications 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. In contrast with these existing methods, we propose a method that is aimed at the orchestrated exploration of the communication and memory system architecture and the computation resource parallelism. Our method 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, as well as, the data parallelism, 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, [11] is not automated, it does not consider the latency of the communication with a shared 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.

The contribution of this paper is the application of our method to an industrial case study: the JPEG encoder. The presented description goes through all the steps of the design flow detailing the used models and the obtained results.

The rest of the paper is organized as follows: Section II presents the Case Study including the description of the application and the customizable hardware platform; Section III describes step by step our customization flow applied to the JPEG encoder, in particular, it introduces the formalism of the application specification and the analytical model of the hardware platform, it describes how the Pareto solutions are selected and used to customize a generic VHDL model of the hardware platform; Section IV discusses the results of the method when applied to a considerable number of applications with different characteristics. Finally, Section V presents the conclusion and future research directions.

II. Case Study

In this Section, the application and the customizable hardware will be described. The application processes data-streams and its design involves constraints on the provided throughput. The customizable hardware architecture includes many processors, each providing a customized data access control, a distributed local storage and data parallel computing units.

A. JPEG encoder

The JPEG encoder [13] is used in multiple multimedia applications. It takes as input raw images and produces as output a compressed bitstream. This example has been chosen as case study because it is representative of streaming applications and is a part of more complex compression standards as MPEG4.

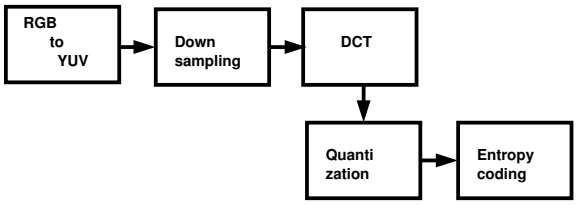


Fig. 1: JPEG encoder

The block diagram of the analyzed JPEG encoder is shown in Fig. 1. It includes several data-stream processing blocks. The first step is a color conversion from RGB to YUV format. After, a downsampling reduces the resolution of the image in the chrominance components and buckets the pixels in macro-blocks of 16×16 or 8×8 for luminance and chrominance respectively. A Discrete Cosine Transform (DCT) converts the YUV into a frequency domain representation. A quantization reduces the number of encoded variations of luminance and chrominance into a number of variations that are perceptible from the human eye. This step corresponds to a frequency low-pass and as a consequence, it increases the homogeneity between the values of the pixels in a macro-block. Successively, an entropy coding compresses the bitstream of data in the macro-blocks by reading data in a zig-zag order, and by applying a run-length encoding (RLE) algorithm, which only encodes the variations of data values.

B. Customizable Hardware Platform

The customizable architecture template includes a library of customizable elements and a set of rules for the element interconnection and customization. An instance of the architecture template is shown in Fig. 2. It contains pipelines ($PIPE_i$ in the figure) of *Processing Tile*s.

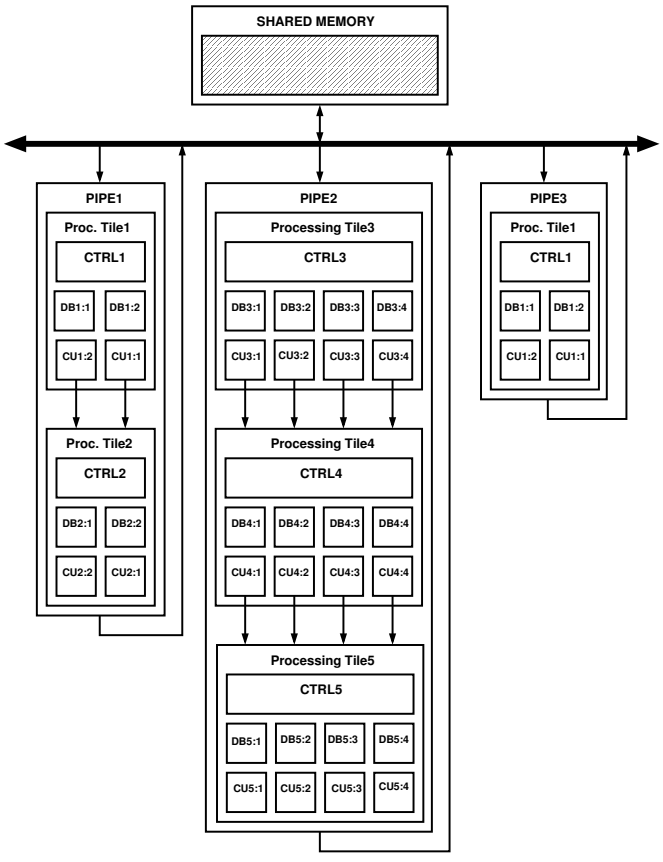


Fig. 2: Customizable architecture template

The *Processing Tile*s within a single PIPE communicate through point-to-point links, while the PIPEs communicate with each other through a shared memory.

Each *Processing Tile* contains blocks of three basic types:

1) a Double Buffering (DB) block, which implements a double buffer mechanism, i.e. a storage point with two buffers alternatively accessed by two concurrent resources (CTRL and CU), one producing and the other consuming data.
2) a controller (CTRL), managing the *Processing Tile* data access and storage from and to the shared memory and to the DB blocks.
3) a computing unit (CU), which implements the computations of an application task core, i.e. a datapath on which repetitions of a loop body execute.

These three block types can be entirely customized for a given application with respect to the data width, the data parallelism, the size of local memories and the realized functionality.

LISTING 1 presents the HDL code of a generic `Processing_tile` which has generic, customizable parameters (lines 2-5), i.e. data width, data parallelism, etc.

```
1  entity Processing_tile is
2    generic(
3      width: integer := 8;   // data width
4      n    : integer := 4;   // data parallelism
5      depth: integer := 32;  // local memory depth
6      ad   : integer := 5    // address width      );
7    Port (
8      clk      : in  std_logic ;
9      init     : in  std_logic ;
10     send     : out std_logic ;
11     data_in  : in  std_logic_vector (n*width-1 downto 0);
12     data_out : out std_logic_vector (n*width-1 downto 0) );
13 end processor1;
14
15 architecture Behavioral of processor1 is
16
17 -- Components Declaration
18 component CTRL generic(...); port (...); end component;
19 component DB generic (...); port (..., read_ad, write_ad,
20                                    data_in, data_out ); end component;
21 component CU generic (...); port (..., data_in, data_out ); end component;
22
23 --signals declaration
24  ...
25 signal T_read_addr : std_logic_vector (n*ad-1 downto 0);
26 signal T_write_addr : std_logic_vector (ad-1 downto 0);
27 signal T_data : std_logic_vector (n*width-1 downto 0);
28 signal T_data1 : std_logic_vector (n*width-1 downto 0);
29
30 begin
31
32 -- Instantiate the components
33 data_out<=T_data1;
34 send<=send_out;
35 ctrl_seq : CTRL generic map (...) port map (...);
36
37 par_store : for i in 0 to n-1 generate
38   par_store : DB generic map (...) port map(...
39   T_read_addr((i+1)*ad-1 downto i*ad) ,
40   T_write_addr, data_in( (i+1)*width-1 downto i*width ),
41   T_data( (i+1)*width-1 downto i*width ));
42 end generate par_store;
43
44 par_compute:for i in 0 to number-1 generate
45   par_compute: CU generic map(...) port map(...
46   T_data( (i+1)*width-1 downto i*width ),
47   T_data1( (i+1)*width-1 downto i*width );
48 end generate par_compute;
49
50 end Behavioral;
```

LISTING 1: VHDL code of a generic Processing Tile

Such a VHDL *Processing Tile* model instantiates the correct number of required parallel resources by combining the concurrent `Generate` statements with `Generate` looping capabilities (lines 37-48). In order to minimize the area overhead due to data access management, a single CTRL is used to spread data among the *Processing Tiles* parallel resources (line 37). The data distribution to the *Processing Tile* parallel resources can be customized through an adequate address computation (line 39-41 and 46-47).

III. HARDWARE CUSTOMIZATION FLOW

Our method aims at analyzing, restructuring and parallelizing a data-oriented application specification and, from this, at:

- infering the customization parameters needed to construct application-specific types of *Processing Tile*'s basic blocks: `CTRL`, `DB` and `CU`.
- selecting the *Processing Tile* customization parameters such as data width, data parallelism, etc. needed to construct one or more required types of *Processing Tiles*.
- selecting a system macro-architecture organizing the *Processing Tiles* in order to realize task pipelines and orchestrate the processor tile communications in locally distributed and globally shared interconnections.

As shown in Fig. 3, our method explores several different loop transformations for an application with *multiple communicating nested loops*.

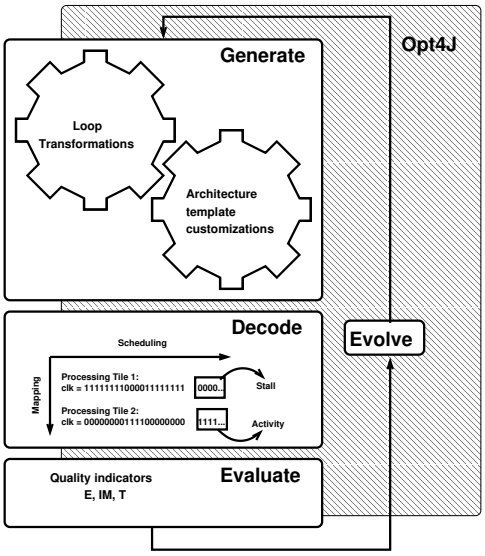


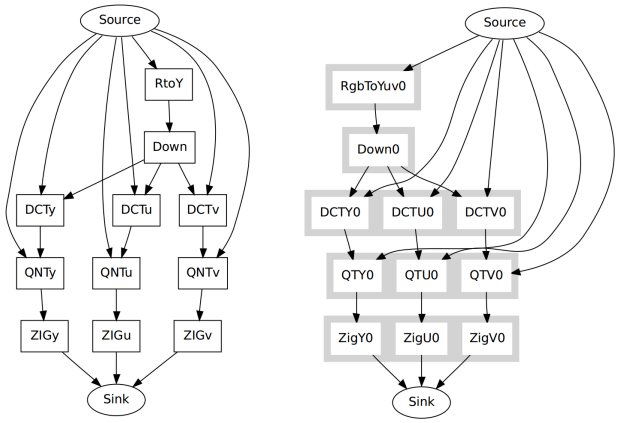
Fig. 3: Cusomization Flow.

From these transformations, such as tiling, paving change and fusion, our method infers some promising *architecture template customizations* as will be explained in Section III-B. The selected hardware solutions are represented by *abstract clocks* [14], [15], which are used to capture the scheduling and mapping information of the iterative tasks of an application. The abstract clocks are binary words marking *Processing Tiles* activities and stalls with 1's and 0's respectively. The abstract clocks also capture the scheduling and mapping modifications due to the loop transformations. They are also used to compute the values of some *quality indicators* such as internal memory

size (IM), energy consumption (E) and system throughput (T), assessing the exploration results. The solution exploration and selection is performed through a genetic algorithm (GA) implemented on the top of Opt4J [16] framework. In our implementation, 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. Opt4J provides a GA backbone that ensures the evolution of an analyzed population towards Pareto solutions evaluated in a multi-objectives optimization search.

A. Application Model

The input specification of the customization flow is given in an Array Oriented Language (ARRAY-OL) [17] that specifies all the information on task and data dependencies of the application needed for the application specific hardware synthesis. Indeed, an ARRAY-OL model instance contains two levels of abstractions for the application description. A higher level (see Fig. 4a and Fig. 4b) specifies task dependencies similarly to a SDF model [18], where tasks consume and produce data tokens at a certain rate and with constraints of output production conditioned by the input availability. At this level, the Array-OL model can be used for a static scheduling and mapping analysis, by using, for instance, abstract clocks.



(a) **Flat** (b) **Hierarchical**

Fig. 4: JPEG Encoder specifications. The white boxes represent flat tasks and the gray boxes represent hierarchycal tasks.

At a lower level, i.e. intra task level (see Fig. 5), an ARRAY-OL model instance captures details about data accesses and data dependencies of a task and expresses them in a matrix form equivalent to a polyhedral model. At this level, parallel versions of the loop nests and their interactions can be explored.

In our flow, the ARRAY-OL specification is given in a textual form (see LISTING 2). For each task, equivalent to a loop nest, it is mandatory to specify the task name, the number of repetitions of the task, and the polyhedral model of each input and output array accessed in the loop body (see Fig. 5).

Tasks can be hierarchical. For instance, in Fig. 4b, a hierarchical specification of the JPEG encoder is given. However, as

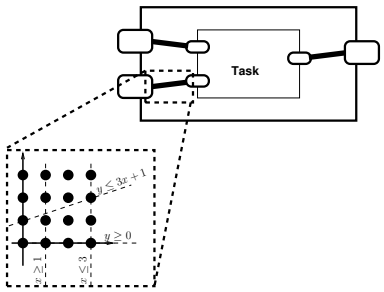


Fig. 5: Intratask information. The polyhedral model is given by the system of inequalities characterizing the complex envelop of the points that represent the data accesses in the iteration space.

also demonstrated by following results, a hierarchical specification of the application prevents some code transformations to be performed, e.g. the fusion of two sub-tasks belonging to two different hierarchical tasks, as for instance `DCTY0` and `QTY0` in Fig. 4b. Consequently, the hierarchy can be used to limit the number and kind of loop transformations and thus to bound the exploration complexity or prevent unwanted hardware instantiations corresponding to specific loop transformations.

```
TN Task_name
TR {100,100} #Task repetitions

#For each Input Dependency
  IDN Input_Dependency_Name
    #Polyhedral Model ...
#For each Output Dependency
  ODN Output_Dependency_Name
    #Polyhedral Model ...

hierarchy {sub_task_name0,...}

LINKS <Source,
       Input_Dependency_name>
       ...
```

LISTING 2: Textual specification

The input application specification also includes Source and Sink nodes to ease the static analysis and scheduling.

B. Application parallelization, mapping and scheduling

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. 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 aforementioned loop transformations 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 communications between them. They also benefit from

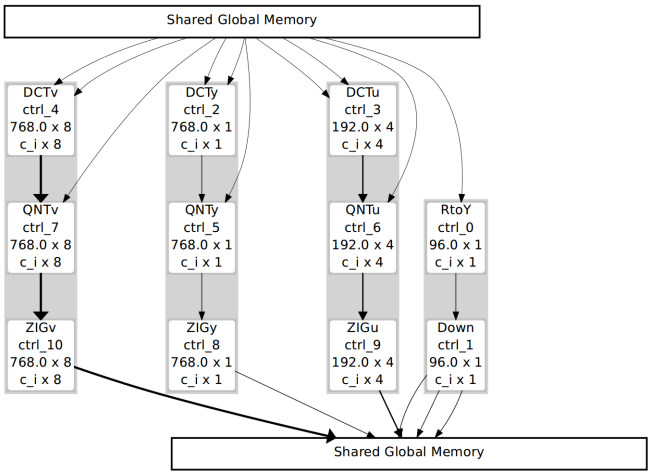


Fig. 6: Example of a structure generated from the exploration of a flat JPEG model. The gray boxes encircle pipelined tasks. The white boxes give information on each pipelined task.

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 and global memory with exclusive read and write memory access modes. 2) The paving-change is used to explore different sizes of the local double buffers. 3) The tiling (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*.

Fig. 6 gives an example of the appliance of loop transformations to the model of a flat JPEG encoder, as specified in Fig. 4a. In such a structure, three pipelines of tasks (DCT, QUT, ZiG) are identified for the luminance (Y) and chrominance (U and V) components. Fig. 6 also shows other important parameters generated for the subsequent hardware configuration such as the selected tiling (and unrolling) factor, indicated as a multiplication factor $\times$k in Fig. 6, and the selected data granularity, indicated as the multiplicand of the tiling factor. For instance, task $DCTv$ in Fig. 6 has a tiling factor of $\times 8$ and a data granularity of 768. These pieces of information are used to customize the VHDL generic model of Fig. 6 by setting the data parallelism and the local memory depth respectively.

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, which give the rythm of data consumption and production for each task, as well as, the synchronization between tasks. The task activities, i.e. data computation and production are marked by 1's, while the synchronization instants are marked by 0's.

Several extensions of the method are under development, as, for instance, the introduction of new mapping rules capable of assigning more application tasks to a same *Processing Tile* or the exploration of design using single buffers for local data storage instead of double buffers. This last modification will force the sequentialization of the data access and computation in a single *Processing Tile*.

C. Analytical Hardware Model

From the abstract clocks, it is possible to compute three indicators of the application restructuring and related MPSoC quality in order to assess the obtained solutions and guide the design exploration.

Such quality indicators are:

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 abstract clock associated with 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 an abstract clock associated with two or more 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 shared 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 the 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 shared memory. In our explorations, we use 0.35 μm SRAM. The size EM of the shared 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 and on the used technology. 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 $b_w = 3.65 \times 10^{-2}$.

D. Multi-objective Exploration and Pareto Solutions Selection

A JPEG encoder exploration has been run by setting parameters that guide the exploration as specified in Table I. For instance the tiling and paving change factors (shortly called loop transformation factors) determine the loop transformations applied to the application specification model and the consequent hardware customizations as specified in Section

GA	
p_1: population size	100
p_2 number of explored generations	100
p_3: number of parents per generation	25
p_4: number of children per generation	25
p_5: parent crossover rate	95%
Other Parameters	
Tiling factors	$\{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$
Paving change factor	$\{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$

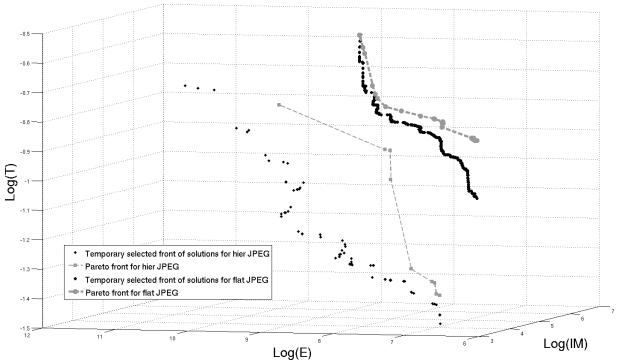
TABLE I: Parameter setting for JPEG encoder exploration.

III-B. Other parameters specify the complexity and precision of the GA exploration.

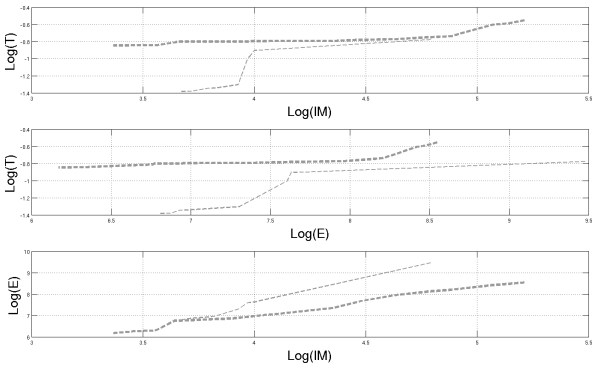
The explorations with the setting of Table I have been performed for the JPEG encoder specifications of Fig. 4a and Fig. 4b. The run-time of the explorations are of 106 and 46 seconds, respectively, as expected in confirmation of the fact that the hierarchy usage lowers the analysis complexity and, therefore, reduces the exploration time. The results of the Pareto front selection are given in Fig. 7. In particular, Fig. 7a shows a 3-Dimensional plot of the two selected Pareto fronts. The thicker line, on the top-right corner of Fig. 7a, shows the results of the exploration for the flat JPEG encoder of Fig. 4a. The thiner line below shows the results of the exploration for the hierarchical JPEG encoder of Fig. 4b. The front with scatter points correspond to a temporary selected front of solutions. The dotted line highlights the final Pareto front selection. Fig. 7b shows the trend of the design quality indicators in 2-Dimensional plots. The plots show the relation between the Throughput (Log10(T)) and the Double Buffers size (Log10(IM)) or the Throughput (Log10(T)) and the Data Access Energy consumption (Log10(E)). The thicker plots show the exploration results for specification of Fig. 4a. The thiner plots show the results for the specification of Fig. 4b. From these results, one can infer that, for the flat JPEG encoder specification (Fig. 4a) our method can achieve better results, which clearly dominate the results of the hierarchical specification (Fig. 4b) with respect to the three exploration objectives, i.e. throughput, internal memory area and energy consumption. The reason of such better results, is the fact that the flat specification does not prevent any loop transformations to be performed and give our method the possibility to substantially explore and restructure the application without many user-specified constraints.

E. VHDL Model Customization and Simulation

We have implemented a JPEG structure (i.e. an architecture with a parallelism and a data granularity) as represented in Fig. 6, on a FPGA Virtex-6 XC6VCX75T. The architecture has been synthesized, placed and routed with ISE Xilinx 13.2 with a speed grade of -2 and a user clock period of $20ns$. The computing units, realizing the DCT, the quantization, etc. are taken from the JPEG encoder of OpenCores [19]. As a comparing example, we have also implemented the OpenCores



(a) **3-Dimensional plot of the two selected Pareto fronts. The thicker line, on the top-right corner of Fig. 7a, shows the results of the exploration for the flat JPEG encoder of Fig. 4a. The thiner line below shows the results of the exploration for the hierarchical JPEG encoder of Fig. 4b. The front with scatter points correspond to a temporary selected front of solutions. The dotted line highlights the final Pareto front selection.**



(b) **Trend of design quality indicators in 2-Dimensional plots. The plots show the relation between Throughput (Log10(T)) and Double Buffer size (Log10(IM)) or Throughput (Log10(T)) and Data Access Energy consumption (Log10(E)). The thicker plots show the trend for specification of Fig. 4a. The thiner plots show the trend for the specification of Fig. 4b.**

Fig. 7: Pareto Fronts. T indicates the throughput in data per cycle, E indicates the energy consumption due to the shared memory accesses in Joule and IM indicates the cumulative size of the local double buffers in number of data (of 8 bits for the JPEG encoder).

JPEG encoder on the same FPGA and with the same synthesis tool setting. The result of our method has a maximum frequency f_{max} of $150Mhz$ with respect to the $97Mhz$ of the OpenCores solution. Our solution can process up to 162 frames of 640×480 pixels per second (Fps) with respect to 136 Fps of the OpenCores solution. The main reason of the throughput and maximum frequency increase for our solution is the usage of a higher degree of parallelism and pipelining, achieved instantiating several times basic computing units (DCT, quantization, etc.) and several local buffers, according to the architecture template of Fig. 2. As a consequence, our solution is 6× larger than the one of OpenCores. However, this area occupancy can be dramatically reduced when identifying hardware sharing possibilities between the the luminance and chrominance pipelines (including DCT, quantization and run-length tasks). Indeed, these pipelines are never activated at

the same time because they share the communication with the Global Memory. Therefore it is possible to merge their hardware datapaths and share the same hardware for their execution. Hardware sharing identification is one of our major future research direction.

IV. Further Experiments with the Proposed Method

This section discusses the experimental results of our GA-based exploration method when applied to several applications having different characteristics. First, the experimental setup is briefly described. Then, the exploration results are reported and discussed.

A. Experimental Setup

In order to illustrate our method, several applications from different domains are processed and the obtained results are discussed. Table II lists the selected applications and provides a brief description of them. The list of selected applications includes applications (e.g. 3mm, 2mm, gemver, fdtd-2d) from PolyBench 3.1 [20] and some other applications (e.g. LPSF, YJPEG) from image processing and multimedia domains.

Application Name	Description
3mm	linear-algebra kernel
2mm	linear-algebra kernel
gemver	linear-algebra kernel
fdtd-2d	stencil application
LPSF	low pass spatial filter
YJPEG	gray-scale JPEG encoder

TABLE II: application descriptions

The explorations are performed on a Intel (R) Core (TM) i5 quad core (2.53GHz) processor running a GNU/Linux operating system with 3.5GB of RAM memory. This analysis compares the results of our GA-based method with those of an exhaustive search that we have developed for comparing reasons. Since the computation power of the host on which the explorations were run is limited with respect to the heap memory size, it is required to adapt the setting of the parameters of the exhaustive exploration to reduce its complexity (e.g. it is required to reduce the size of the explored population for complex problems). As a consequence of this limitation, and to keep valid the comparison between the exhaustive and GA-based explorations, we also limit the exploration capability of our GA-based method even though this is not required. Indeed, with the available computing resources, the computation complexity of our exploration allows performing very complex explorations within few seconds. Consequently, the parameters $(p_1, p_2, p_3, p_4, p_5)$ of the genetic algorithm are set to the values provided in Table III. Additionally, table III gives information on the loop transformation factors (k) for each corresponding application.

B. Exploration Results

Table IV reports the complexity of the performed explorations by reporting for each analyzed application: the number of tasks in the analyzed application, the number of possible

	$\{p_1, p_2, p_3, p_4, p_5\}$	k
2mm, fdtd-2d,gemver, LPSF	{40, 10, 4, 4, 95%}	{1,2,4}
3mm	{40, 10, 4, 2, 95%}	{1,2}
YJPEG	{10, 5, 4, 4, 95%}	{1,2}

TABLE III: Opt4J parameter setting: p_1: population size, p_2: number of explored generations, p_3: number of parents elected for reproduction (i.e. crossover), p_4: number of children elected to integrate the new generation and p_5: crossover rate

fusions, the total number of explored individuals, the number of possible individuals in the whole exploration space and the number of selected Pareto solutions. For example, *3mm linear-algebra* application has 6 tasks and 15 possible task fusions. For each fusion, the initial population of the genetic algorithm includes 40 individuals and it evolves for 10 generations. At each evolution, 2 new individuals are created and explored. Considering the parameters defined in Tables III and IV, the expression $((p_2 * p_4) + p_1) * c_2$ formulates the total number of explored individuals (c_3). Hence, value of c_3 for 3mm is $((10 * 2) + 40) * 15 = 900$. Furthermore, the formula $k^{c_1} * c_2$,

	c_1	c_2	c_3	c_4	c_5
3mm	6	15	900	960	3
YJPEG	5	15	450	480	4
LPSF	4	8	640	648	1
2mm	4	6	480	486	3
fdtd-2d	4	4	480	486	2
gemver	4	6	480	486	3

TABLE IV: Exploration Complexity and Selectivity: c_1: number of tasks in the application, c_2: number of possible fusions, c_3: total number of explored individuals, c_4: number of possible individuals in the whole exploration space and c_5: number of Pareto solutions.

where k corresponds to the number of explored transformation factors, gives the number of explorable space of individuals (c_4). Therefore, value of c_4 for 3mm is $k^6 * 15 = 960$, where $k = 2$. As it can be observed from the table, c_3 values are very close to c_4, this is because the exploration complexity is kept low. More complex explorations and the formal framework of the GA-based method are presented in [21]. Even though the number of explored individuals is close to the number of possible individuals in the whole space, our method outperforms exhaustive search with respect to exploration run-time (see Table V).

Table V provides the information on the exploration run-time for the GA-based (r_1) and exhaustive ($\overline{r_1}$) methods. Moreover, it gives the percentage of run-time for each exploration step of the GA-based exploration: fusion exploration (r_2) and genetic algorithm in Opt4J (r_3). As it can be seen from the table, most of the time for the GA-based method is spent by the the Opt4J framework to perform genetic algorithms. The r_2 values are relatively small compared to the r_3 values; however r_2 values trend to significantly increase in proportion with the increase of the application complexity (e.g. 3mm, YJPEG).

In order to assess the quality of our method, the output of our method, which is a Pareto-front, is compared to the Pareto-front provided by an exhaustive search. The exhaustive search explores all the possible application transformations in

	run-time (sec.)		indicators)		
	r_1: GA	$\overline{r_1}$: exhaustive	ϵ-indicator	$r_2(\%)$	$r_3(\%)$
3mm	2.2	20.9	1	15	85
YJPEG	3.1	35.6	1.11	19	81
LPSF	2.0	20.5	1	7	93
2mm	2.1	26.8	1.26	6	94
fdtd-2d	2.8	20.7	1	7	93
gemver	4.4	22.0	1	9	91

TABLE V: Exploration Run-Time: r_1: total run-time of the GA-based exploration in seconds, $\overline{r_1}$: total run-time of the exhaustive exploration in seconds, r_2: percentage of the GA-based method run-time spent for fusion exploration, r_3: percentage of the GA-based method run-time spent for GA running.

the specified space. The comparison between our method and the exhaustive search is assessed through the computation of the ϵ-indicator, as presented in [22].

Basically, the ϵ-indicator checks the closeness of two Pareto-fronts, in our case, the reference Pareto-front is output of the exhaustive exploration. The closer the ϵ-indicator is to 1, the better the solution. Table V also provides the ϵ-indicators to measure the closeness of the Pareto fronts of the GA-based and exhaustive methods. This table demonstrates that our method is rapid and efficient in a sense that it explores the design space in a significantly less amount of time compared to the exhaustive exploration. Moreover, our method is effective because it still maintains the Pareto-front solutions, inferred from the fact that the ϵ-indicator values are either 1 or close to 1.

V. Conclusion and Future Works

In this paper, we presented a method for the design of MPSoCs for data-intensive applications. Our method uses loop transformations to perform a blend optimization of the hardware communication structure, the memory hierarchy and the computing resource parallelism. A JPEG Encoder is studied as a representative case study in order to demonstrate all the steps of our method. Furthermore, several applications from different domains are tested in order to demonstrate the method quality and applicability. Experimental results show that the proposed method is rapid and efficient. The JPEG encoder implementation demonstrates that our method allows for implementations with a significant throughput increase, an increase of the supported f_{max} of the FPGA implementation and 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, the memory and the computing architectures. Several future research directions are possible, such as integrating into our method additional relevant loop transformations; enlarging the list of the possible mapping and scheduling rules; exploring hardware sharing and applying our method to the case of Application Specific Instruction Set Processors.

VI. 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] M. Thompson, H. Nikolov, T. Stefanov, A. D. Pimentel, C. Erbas, S. Polstra, and E. F. Deprettere, "A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs," in *CODES+ISSS'07*. New York, NY, USA: ACM, 2007, pp. 9–14.

[3] A. Bakshi, V. K. Prasanna, and Á. Lédeczi, "Milan: A model based integrated simulation framework for desgin of embedded suystems," in *LCTES*, 2001, pp. 82–87.

[4] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. L. Sangiovanni-Vincentelli, "Metropolis: An integrated electronic system design environment," *IEEE Computer*, vol. 36, no. 4, pp. 45–52, 2003.

[5] R. Schreiber, S. Aditya, S. Mahlke, V. Kathail, B. Rau, D. Cronquist, and M. Sivaraman, "Pico-npa: High-level synthesis of nonprogrammable hardware accelerators," *J. of VLSI Signal Proc.*, vol. 31, no. 2, pp. 127–142, Jun 2002.

[6] 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.

[7] 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.

[8] 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.

[9] 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*.

[10] 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.

[11] 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.

[12] 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.

[13] G. K. Wallace, "The jpeg still picture compression standard," *Commun. ACM*, vol. 34, no. 4, pp. 30–44, Apr. 1991. [Online]. Available: http://doi.acm.org/10.1145/103085.103089

[14] 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)*.

[15] R. Corvino and A. Gamatié, "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.

[16] 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)*.

[17] 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.

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

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

[20] PolyBench: The Polyhedral Benchmark Suite. [Online]. Available: http://www.cse.ohio-state.edu/ pouchet/software/polybench/

[21] R. Corvino, A. Gamatié, M. Geilen, and L. Jozwiak, "Design space exploration in application-specific hardware synthesis for multiple communicating nested loops." in *SAMOS 2012*, 2012.

[22] 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.