EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

Parallel embedded processor architecture for FPGA-based image processing using parallel software skeletons

Hanen Chenini[1]*, Jean Pierre Dérutin[1], Romuald Aufrère[2] and Roland Chapuis[1]

Abstract

Today, the problem of designing suitable multiprocessor architecture tailored for a target application field raises the need for a fast and efficient multiprocessor system-on-chip (MPSoC) design environment. Additionally, the implementation of image processing applications on MPSoC system will need to exploit the parallelism and the pipelining in algorithms with the hope of delivering significant reduction in execution times. To take advantage of parallelization on homogeneous MPSoCs and to reduce the programming effort, the proposed design methodology offers more opportunities for accelerating the parallelization of sequential processing image algorithms on pipeline architecture. Our approach provides rapid prototyping tool as a graphic programming environment (CubeGen). Further, it offers a set of parallel software skeletons as a communication library, providing a software abstraction to enable quick implementation of complex image processing applications on field-programmable gate array (FPGA) platform. The design of homogeneous network of communicating processor is presented from the hardware and software specification down to synthesizable hardware description. Then, we extend our approach to support more complex applications by implementing a soft multiprocessor for 'multihypotheses model-driven approach for road recognition' and show the impact of various configuration choices (hardware and software) to match the specific application needs. Using the images of a real road scene, the performance results of the road recognition algorithm on a Xilinx Virtex-6 FPGA platform not only achieve the desired latency but also further improve the tracking performance which depends mainly on the number of hypotheses.

1 Introduction

In recent years, the complexity of embedded systems based on multiprocessor system-on-chip (MPSoC) architectures dedicated to very computationally demanding tasks in particular image processing applications has led to the emergence of new enhanced MPSoC design methodologies. However, as more processing nodes (heterogeneous or homogeneous) are integrated in a single chip, the fitting of computational tasks to hardware resources is still a challenging task since its related to the optimal exploitation of the different types and degrees of parallelism among multiple processing elements available in the MPSoC design. Unfortunately, hardware development tailored to multitasks application is more difficult; generally, it can be performed only by expert users.

Indeed, our general problem requires the rapid hardware prototyping of complex image processing applications (multitasks applications) in embedded devices and multiprocessors design in this case. These latter must be able to satisfy real-time and embedding constraints found especially in the field of intelligent sensor for mobile robots. In real world, sensors are usually embedded on board a moving vehicle, and consequently, the embedded systems are constrained to face the hard real-time constraints imposed by moving vehicle applications. The approach that we have taken in solving these multiple requirements consists in moving the computing architecture to the sensor itself to achieve the necessary processing power to run the algorithms near the camera. We have focused on combining cameras and algorithms to understand the vehicle environment and to provide some

*Correspondence: Hanen.Chenini@etudiant.univ-bpclermont.fr
[1] Institut Pascal-UMR 6602 CNRS, Blaise Pascal University, 24 Avenue des Landais, Clermont-Ferrand 63177, France
Full list of author information is available at the end of the article

intelligent processing directly into the camera. Since the final aim of this work is to integrate these applications on a mobile vehicle, it is necessary to execute these algorithms under several requirements using an architecture with high computation capability, low power consumption, flexibility, low memory, and small size. Thus, the use of field-programmable gate array (FPGA)-based parallel multiprocessor design allows us to manage a large amount of data and work within these multiple requirements. Moreover, it is important to use the minimum required resources to allow the entire system to be integrated on the same chip. This allows the reduction of the SoC complexity and cost.

These issues increase the demand for providing a framework that is able to automatically generate a suitable multiprocessor configuration according to the real-time requirements and the features of a given image processing application. More precisely, our goal is to develop rapid prototyping tools for image processing applications, using parallel homogeneous architecture, as will be described in the next section. In response to that, we have proposed a new MPSoC approach [1] that aims at raising the abstraction level of the specifications for both software and hardware providing the necessary tools supporting the design from the specification down to the embedded implementation. In addition, our methodology proposes a complete generic architecture from which code can be generated automatically. Furthermore, our new design methodology is able to support parallelization of complete image processing applications using multiple instruction multiple data processors (MIMD). From a sequential application, the proposed graphic programming environment (CubeGen) leads us to generate MPSoC system implementation on an FPGA with the parallelized application implemented onto it only in few hours. To this end, we build a programming environment dedicated to the fast prototyping of embedded vision applications based on parallel algorithmic skeletons. Due to computationally intensive nature of processing algorithms, the corresponding software code for each processing node is also generated automatically using specific skeleton and its associated communication functions that facilitate the conversion from sequential algorithm to a parallel C code. These tools represent an important step towards simplification of application implementation in FPGA platform. Another important advantage is that the proposed FPGA design flow offers great potentials for quickly making several experiments with different MPSoCs and exploring configuration choices during the design process. Therefore, the development time for applications running on these architectures is easier and faster than for hand-designed architecture. Evidently, the use of such approach makes it possible to adjust the architecture by refining of the material architecture where it is needed to efficiently meet the requirements of a given application. This often implies that the software has to be developed at the same time the hardware architecture is refined to provide a design with enough calculation capacity and flexibility.

In this current work, we focused on real-time image processing algorithm (as object recognition, feature extraction, learning, tracking, 3D vision, etc.) in FPGA suitable for embedded vision systems to develop real-time algorithms that are able to assist the driving activity. In particular, the ability to recognize objects from data acquired by sensors is important for building intelligent systems. Based on the proposed MPSoC methodology, we propose a real-time, embedded vision solution for multihypothesis approach for road recognition implemented on parallel multistage architecture. In fact, the proposed road recognition algorithm-based lane model is able to recognize a wider range of lane structures such as straight, curve, and parabolic models according to various driving environment. In addition, it is robust against shadows, noises, etc. due to the use of the parallel knowledge of road, vehicle, and camera parameters. Hence, this approach relies on roadway sensors to obtain the lateral vehicle position in its lane as well as the steer angle and the road curvature. The lane detection problem is formulated by determining the set of localization parameters. In addition to its ability of road detection, our architecture is able to recognize the roadsides by taking into account the coherence of the two sides of the road which is performed by a flexible interconnection network.

The rest of this paper is organized as follows. We briefly present in Section 2 the state of some design methodologies that are able to automate the hardware implementation of an application algorithm on MPSoC architecture. We describe in Section 3 the proposed fast prototyping methodology which helps the software and hardware designer to obtain quickly an efficient implementation of this application on HNCP architecture. In Section 4, we use a motivating example of image processing application multihypotheses recognition approach to illustrate the proposed design flow. Section 5 describes the parallel structure of the algorithm and the choices made while porting the application software to the embedded system. We show in Section 6 the results of the implementation of the road recognition into Xilinx FPGA following the proposed design methodology. Section 7 concludes the paper.

2 Related work

Creating an embedded multiprocessor system which meets its performance, cost, and design time goals is a hardware-software co-design problem where the design of the hardware and software components influences each other. Actually, co-design [2] is usually used as the design method for embedded systems. For example, the co-design approach MoPCoM [3] allows the use of generic unified modeling language(UML) tools based on the modeling and analysis of real-time and embedded systems (MARTE) standard [4]. In fact, the MoPCoM methodology is performed using three abstraction levels: abstract, execution, and detailed modeling levels (AML, EML, and DML, respectively).Whereas, we have to respect multiple design rules to build UML models for embedded systems in order to perform automatically VHDL (VHSIC hardware description language) code generation. In addition, for MPSoC cases, the multicomponent architecture raises problems in terms of application distribution related to the manual data transfers and synchronizations which become very complex and result in loss of time and potential deadlocks. Subsequently, for programming these architectures efficiently using co-design approaches, hardware description languages (HDL), such as Verilog or VHDL, are required. Unfortunately, the users especially the programmers work with high-level programming languages (C, C++), and they do not know how to deal with hardware description languages. Hence, they have to focus on low-level system issues, and they have to manage low-level communication problems such as deadlocks and parallelism details which require a technical background rarely found among non-expert users. Consequently, designing an interconnection architecture for MP-SoCs while simultaneously reducing power consumption and satisfying the performance constraints is a difficult problem.

One suitable design process solution consists of using rapid prototyping methodology. The aim is then to go from a high-level description of the application to its real-time implementation on target architecture as automatically as possible. This automation saves development time and prevents conflicts and deadlocks. In this way, some works have shown that the major challenges of rapid prototyping image processing applications in multiprocessor system-on-chip design are mainly the programming model, the design flow, and the communication infrastructure. Addressing successfully these issues would extremely enhance the performance of the computing architectures in terms of power consumption, design cost, faster execution times, and shorter development cycles. Recently, many research works proposed a high-level programming environment for designing MPSoC architecture and testing complex image processing systems. An example of rapid prototyping methodology based on the SynDEx tool is presented in [5]. This work presents the use of this rapid prototyping methodology suitable for image processing systems and heterogeneous multicomponent architectures. The SynDEx tool generates synchronized distributed executives from both application and target architecture description models. To illustrate their methodology, experimental results are presented using a complex multilayer application including video and digital communication layers, going from its high-level description to its distributed and real-time implementations on heterogeneous platforms. Unfortunately, for designers of such embedded architectures, the challenge is to find an optimal configuration for several units (including processors, memories, communication devices, etc.) and optimize the interconnections among heterogeneous processing units and memories. Consequently, they cannot react rapidly to satisfy the user's needs and to increase the performance/power consumption ratio of these embedded systems.

Other methodologies provide functionalities that improve or ease the software/hardware specification and the hardware implementation of algorithm application onto multiprocessor architecture. These design methods used often a high-level description (for example, in SystemC) such as ROSES [6] and GRACE++ [7] and therefore depends heavily on synthesis tools to optimize the design. When using this high-level component-based approach, the overall experiment took about 4 months (not counting the effort to develop library elements and debug design tools). Running all wrapper generation tools, the designer has to write the virtual architecture model with all the necessary configuration parameters. Other examples of related work Koski MPSoC design flow and SystemC-based design methodology can be found in [8] and [9], respectively. Koski provides a single infrastructure for modeling of applications, automatic architectural design space exploration, and automatic system-level synthesis, programming, and prototyping of selected MPSoCs. The methodology in [9] supports automated design space exploration, performance evaluation, and automatic platform-based system generation. The main disadvantage of these two methods is the need for automated parallelization of applications and design space exploration at application level. Moreover, these works require applications to be specified by hand in UML or SystemC. Early interesting Daedalus 'system-level design for multiprocessor system-on-chip' framework [10] offers a fully integrated tool flow in design space exploration (DSE), system-level synthesis, application mapping, and system prototyping of MPSoCs architectures. It provides the parallelization of the application through KPNgen, the architecture exploration through SESAM, and the synthesis of the system through ESPAM. Whereas, their MPSoC system performance was limited by the available on-chip FPGA memory resources and the available intellectual property (IP) cores in Daedalus RTL library. In [11], the authors have developed a high-level design methodology for FPGA-based multiprocessor architecture. This methodology enables the architecture development and the application mapping onto an FPGA-based runtime reconfigurable multiprocessor accelerator. They proposed a tool based on integer linear programming to explore the micro-architectures and allocate application tasks to maximize throughput. Using this tool, we implement a soft multiprocessor for IPv4 packet forwarding that achieves a throughput of 2 Gbps. Their exploration framework ignores the arbitration overhead when computing the communication access time. This can be a significant source of error when there are a large number of masters on a bus. They need to extend the framework to include arbitration overhead to eliminate this source of error.

Our work differs from those cited by focusing especially on matching tasks into the homogeneous-based FPGA system. The particularity of our approach aims at proposing a high-level framework for FPGA-based image processing to provide user guidance and automatic tools for designing and implementing complex image processing applications. Hence, the designers need only to optimize their applications, keeping the same input-output functionality of the software applications and the design of the underlying hardware platform. Given a complete algorithm specification(C or C++), the designer identifies where to introduce parallelism through software skeletons for data and/or task parallelism which are completely different from hardware skeletons. Traditionally, the concept of hardware skeletons has been first presented by [12]. In their study, they have discussed the use of the notion of hardware skeletons, specific to application field which is novel to the hardware domain. They implemented their hardware skeleton library as a hierarchy of three levels of hardware blocks: arithmetic core library, basic image operation library, and finally, high-level (compound) skeleton library. However, this approach employed high-level descriptions of task-specific architectures specifically optimized only for Xilinx XC4000 FPGAs. In this case, the application designers have manually selected the appropriate implementation among different alternative solutions, and also, this approach needs to be extended to support more complex applications (i.e., to support more arithmetic types and providing other skeletons).

Without loss of generality and work with Xilinx or Altera FPGAs, our proposed homogeneous system can be implemented on any FPGA as long as it is large enough. Additionally, the developers will not find problems related to hardware specification or optimization to efficiently embed their applications. Here, the users can easily choose which parameters are needed in their implementation in order to either save memory or meet performance requirements. Additionally, they do not deal with low-level system issues and they have not manage low-level communication problems such as deadlocks and parallelism details.

3 Proposed parallel architecture

3.1 Motivation

Due to the increasing complexity of MPSoC embedded systems and image processing applications, the performance of multicore systems is greatly constrained by the memory wall, the communication wall, and the application complexities wall. Despite the efficiency of heterogeneous system, designers as well as system architects are facing completely new challenges to quickly and efficiently design parallel architectures tailored to complex applications to meet performance, power, and cost goals. Two major problems are (1) hardware language knowledge required to deal with the development of heterogeneous systems (global communication interconnect, interconnect interfaces, complex memory hierarchies, etc.) and (2) the difficulty of programming applications to use diverse computing cores. Since many engineers have knowledge of C prior to program their applications, we suggest the use of homogeneous processors for rapid prototyping of software application coded in high-level language (C or C++). Consequently, in our approach, the challenge is to adapt these algorithms to homogeneous system. Overall system performance can be improved by allowing the homogeneous cores to work collaboratively on different sequential parts of an application and increasing the number of processors to achieve the high computational requirement for such applications. In fact, these applications can be decomposed into a number of subtasks that need to be performed in sequence and with each subtask being executed in a dedicated hardware architecture that will operate concurrently with all other subtasks. With the management of the parallelism in these applications, the user can further distinguish the algorithms by the type of parallelism (task or/and data) that takes into account its structure to propose efficient solutions with task-parallel or/and data-parallel computations in a homogeneous architecture. A way to expose and exploit increased parallelism, to in turn achieve higher scalability and performance, is to write parallel applications that use both task and data parallelism. In the face of real-time requirements, we have designed a pipeline architecture which offers parallel processing capability for more efficient at executing the complex algorithms used for intensive tasks. Indeed, the developed homogenous multistage architectures are well suited to the computational needs and real-time performance of multitask real-time application requiring much parallelism. The use of pipelining improves the performance as compared to the traditional sequential execution of tasks. By exploiting efficiently the processing power, the execution of algorithms will be balanced between the parallel computing stages, which will improve greatly the overall performance. Moreover, the proposed architecture is composed of a set of parallel computing stages connected in serial where each stage contains an hypercube of processors. Each stage holds several computational nodes connecting through communication links and running in parallel, each processing part of an application (single task). In addition, two consecutive stages are connected via unidirectional first in, first out (FIFOs) in charge of control and synchronization. Indeed, designing an image and video processing system can be complex and time-consuming; as a result, the design process can take months depending on the system's complexity. The adopted method developed to ease this bottleneck is to create a graphical tool called 'CubeGen' which automatically generates the hardware for a given application. In practice, with this framework, the system is built entirely hiding unnecessary hardware complexities and you can only focus on optimizing the sequential code and investigating improvements of the parallel code. The user does not need to know anything about hardware language; its application can be designed in C language and implemented automatically into our hardware architecture. For rapid prototyping of complex sequential applications, we address these issues by providing:

- Hardware architectures supporting the simultaneous use of data parallelism and task parallelism. Communication at each pipeline stage is mostly depending in the type of parallelism (for example for static data or task parallelism, communication point-to-point links are required).
- Parallel programming libraries enable developers to accelerate applications. Depending on the structure of the application (i.e., given the sequential code of each subtask), the user can generate the parallel application code by deciding which is the appropriate skeleton from a set of parallel skeletons making the basis of our programming environment (split, compute, and merge (SCM) skeleton, FARM for data/task parallel processing, and PIPE for task-parallel parallelism, split, compute, communication all to all and Merge (SCComCM), etc.).
- New design framework 'CubeGen' targeting FPGA-based image and video processing applications with the purpose of accelerating the development time by utilizing a set of parallel software skeletons and pre-built hardware blocks.

In the following, we describe the architecture including hardware and software aspects, highlighting the automatic generation of homogeneous network of communicating processor(HNCP).

3.2 Pipelined homogeneous design

Figure 1 illustrates the concept of the pipelined architecture. Each stage of the pipeline consists of 2^D nodes, each containing a soft processor P_{i_j}, with private memory (Mem) for application software and data storage, and a communication device (Comm) (simple FIFO point-to-point (FSL link) communication, or more complex communication such as direct memory access (DMA) router packet). The arrangement of the various element nodes in each stage is a static network with regular hypercube topology.

We have defined a set of rules to build a multistage homogeneous architecture, from which hardware code is automatically generated by means of CubeGen framework. Suppose now that we have a pipeline architecture with K pipeline stages which work in parallel, each stage comprises $N_i = 2^{D_i}$ nodes ($i \in [1, K]$). Here, we define two different types of multistage architecture as illustrated in Figures 2 and 3. Taken two consecutive pipeline stages (i and $i + 1$), if the cube nodes in each stage are connected by bi-directional communication links (point-to-point connections), each processor P_{i_j} in the stage i will communicate with the processor P_{i+1_j} in the next stage via point-to-point fast simplex link (FSL) which is unidirectional (for $j \in [0, N-1]$ where $N = \min(N_i, N_{i+1})$ and N_i and N_{i+1} are the number of processors in the stage i and $i + 1$, respectively). Consequently, each processor in a given stage can send their output to the processors in the next stage, but they cannot receive from these processors. Figure 2 illustrates this point, a multistage architecture which contains three consecutive stages based on bidirectional point-to-point FSL, with 16 processors in the first stage, 8 processors in the second stage, and only 4 processors in the last stage. Whereas, if we have two consecutive stages where the first is based on point-to-point (FSL) links and the second is based on hardware router, only processor P_{i_0} in the stage i will communicate with the processor P_{i+1_0} in the stage $i+1$ (only the processors with index 0 are connected via unidirectional link (Figure 3)).

The processing latency on parallel applications in such pipeline architecture depends on the execution time of the slowest pipeline stage. Consequently, the processing time for the proposed pipeline architecture is equal to the execution time of the first stage t_{stage_1} or the execution time of the second stage t_{stage_2} whichever is the maximum $\max(t_{\text{stage}_1}, t_{\text{stage}_2})$. To boost the performance of the application, CubeGen framework is able to generate multiple pipeline stages with different network dimensions, communication links, software skeletons, etc. Of course, this has an effect on the performance of each individual stage; further, it can improve significantly the overall performance of the complete design.

3.3 Hardware-software video streaming modules

In practice, image and video processing applications require large amount of data transfers between the input and output of a system. For example, a 256 × 256 color image has a size of 196,608 bytes. The penalty in

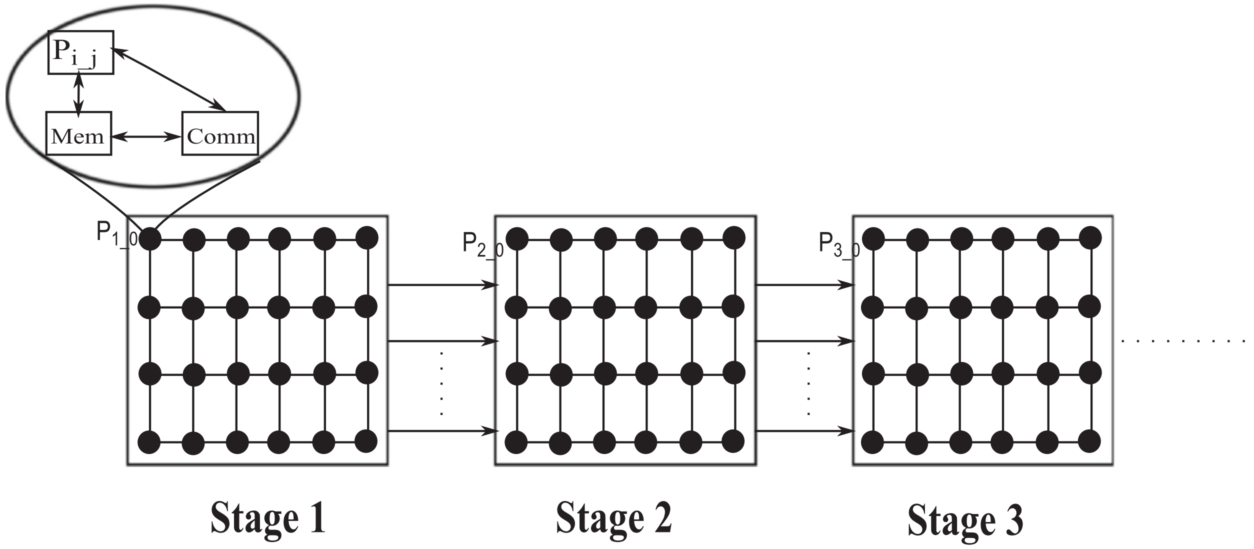


Figure 1 **Homogeneous parallel processor architecture.**

performance for these applications comes from the high latency communication between the different processing units. The image acquisition blocks play a vital role of capturing the incoming video and thus, determine the system's overall performance. In fact, MPSoC-based systems are for the majority of the time unsuitable under real-time constraints. They cannot achieve the required high performance that is expected when working with video frame rates. Therefore, dedicated embedded architectures need to be designed. In our approach, video data are captured from a PC using the Ethernet input port with frame resolutions up to 256 × 256 at frequency fixed by the user. Then, these video data need to be stored in memory, transferred to processing nodes, and displayed on the monitor through the VGA output port after processing it. With dedicated video input device design offering parallel processing capability for video applications, we have built a flexible architecture that enables the user to perform real-time processing on a single frame or multiple frames.

3.3.1 *Ethernet video module*

The proposed IP Ethernet module receives the data coming from the FPGA Ethernet port. Therefore, the developed IP employs the TriMode Ethernet MAC hardware block available in the utilized FPGA. In our system, we have used the reduced gigabit media independent interface (RGMII) to realize the communication between Ethernet MAC and PHY (the block PHY (physical) implements the Ethernet physical layer of OSI network model). For management data input/output (MDIO) ports on the Virtex-6 Embedded Tri-Mode Ethernet MAC (V6EMAC) are used to access the registers in the internal and external PHY. The proposed module is configured through a set of Ethernet-accessible registers. The reliability and efficiency

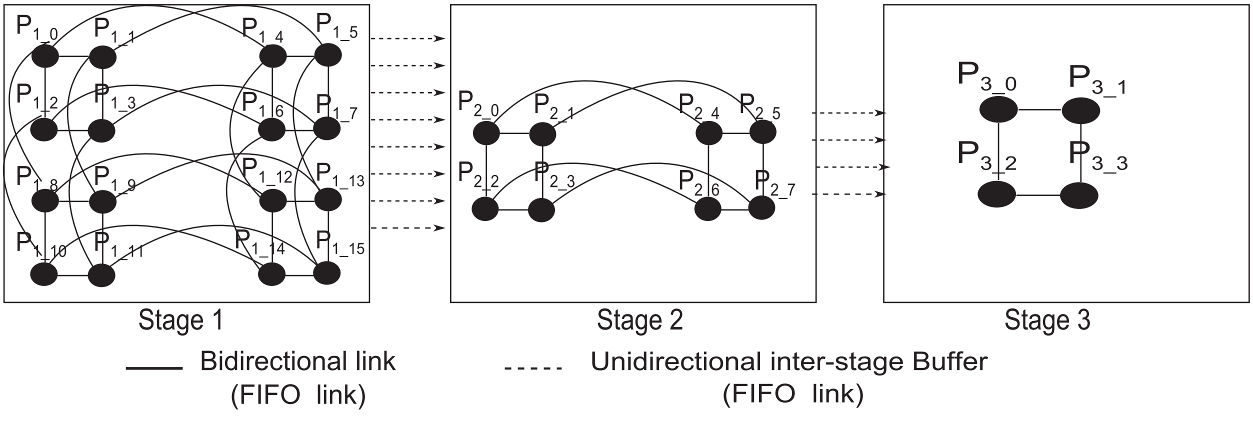


Figure 2 **Hardware architecture with three pipeline stages (connected through point-to-point connections).**

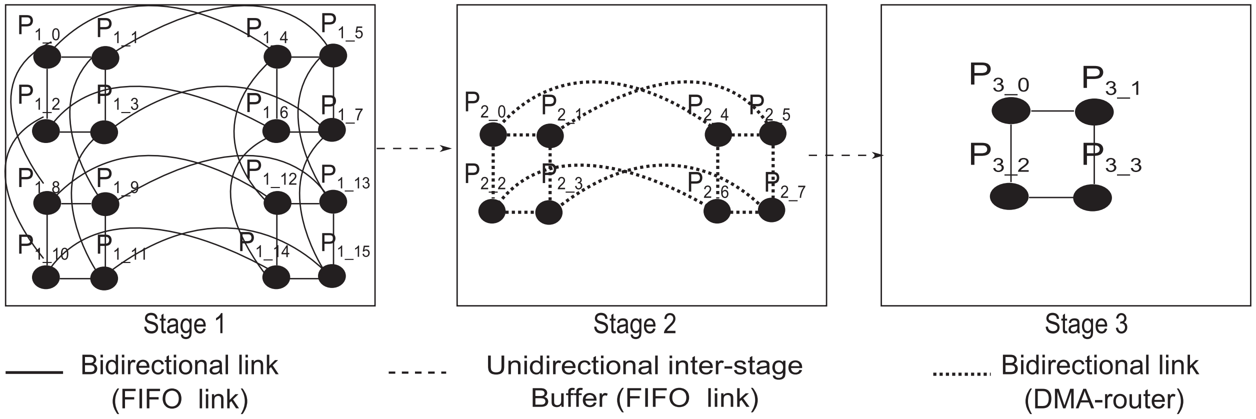


Figure 3 Three-stage pipelined architecture (connected through point-to-point connections and hardware router IPs).

is ensured by data completeness test and retransmission mechanism. It is intended that this design can be quickly adapted and downloaded onto an FPGA to provide a hardware test environment.

3.3.2 Bus video module

To send the video source to a large number of computing nodes, we have proposed video input device composed of frame generator with multiple frame grabbers (Figure 4). In fact, we use this design when only a portion (window) of input image needs to be captured by the processing nodes for processing. In practice, the developed frame generator module receives the output signals from the IP Ethernet module seen previously via point-to-point connections (FSL links) and transfers the input image to one or more frame grabber modules. The received pixels (each pixel is coded in 8-bit gray level) are collected in local memories of size 64 kB of block RAM (random access memory) (BRAM) associated to the soft processors. Each local memory stores at least one complete frame and is used if the bandwidth of the bus is not small to transport the digitized video data stream without loss. The maximum size of each received frame is 256 × 256 pixels, and the time between two successive images is fixed by the user when he/she sends the frame via Ethernet port. Thereafter, it is possible to collect the digitized video data stream directly in the main memory (e.g., the input buffer) of each frame grabber module associated to processing nodes. Thankfully, these grabbers used sync signals to control the synchronization pulses between the input (incoming data from video bus) and the output buffer (outgoing data to processing nodes). The vertical swap

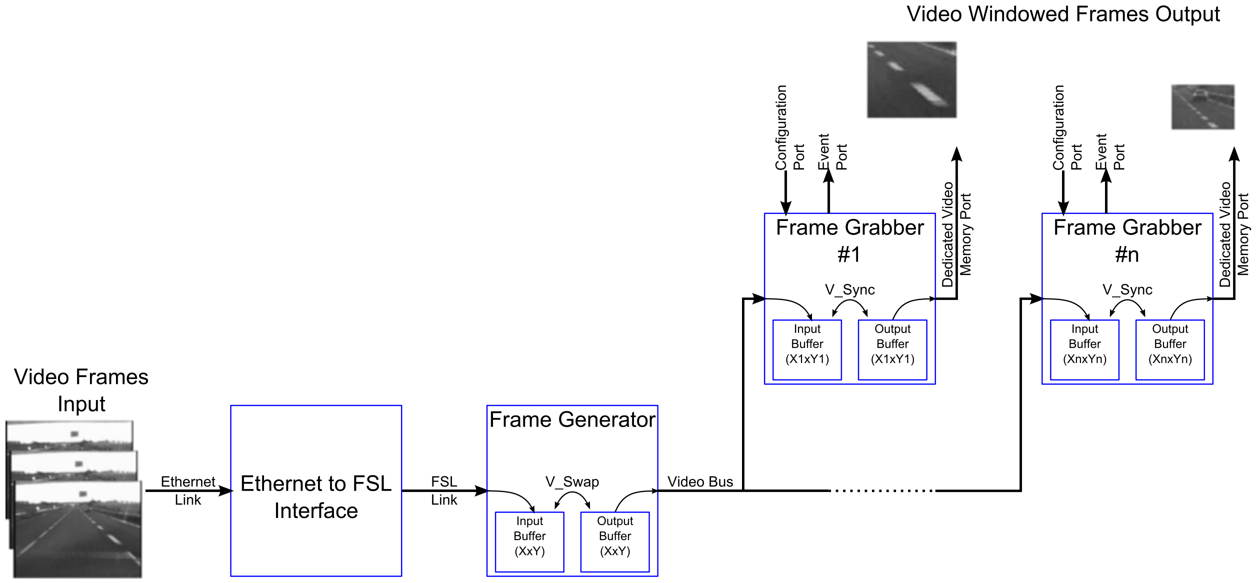


Figure 4 I/O architecture-based frame grabbers.

(V-Swap) indicates the beginning of new frame; the horizontal sync (H-Sync) indicates the last pixel frame line, whereas the vertical sync (V-Sync) indicates the last pixel of each frame.

3.4 Algorithmic skeletons

To embed the software application written as serial computation into the MPSoC design, there are many problems related to the decomposition, distribution, code and data sharing, communication and synchronization, etc. To this end, we have developed a set of common and recurrent forms of parallelism algorithmic skeletons corresponding to static data parallelism, dynamic data parallelism, task parallelism, or flow parallelism in order to implement a sequential algorithm onto our MPSoC architectures. In this way, we have proposed a set of high-level communication functions to easier prototype the target application and to make them readily available for the application programmer. To help the designer, the communication functions are automatically generated by the CubeGen framework for each dimension of the MPSOC system. This makes it much easier for a developer to produce a parallel program tuned for the developed architectures.

3.4.1 SCM skeleton

This skeleton is based on a static geometric decomposition (row-based, column-based, or rectangular subimages) over data where the data is divided up into a number of equal-sized parts, and each part is processed by a different worker (Figure 5). In a basic case, the split function is just a synchronization barrier (all processing nodes have their data subsets) but also can contain a preprocessing function and/or a send function of each data subset. Moreover, the processes of distributing the input data and gathering the output data need to be included in the skeleton's definition, done by the same processing node. Some other skeletons are derived from this skeleton such as SCComCM (Split, Compute, Communication all-to-all,Compute, Merge) and SCComMC (Split, Compute, Communication all-to-all, Merge, Compute) which are introduced to the parallel implementation of multilayer neural network [13] and dynamic neural field (DNF), respectively. Nevertheless, some image processing applications require irregular data set processing, for instance, an arbitrary list of different sizes of windows changing at each iteration (image). In this case, a static allocation of computing processes is not always suitable because of an uneven workload between processors.

3.4.2 FARM skeleton

This skeleton shows its utility when the application requires the processing of irregular data, for instance an arbitrary list of windows of different sizes (Figure 6). This method is only efficient, however, if there are more data elements than processors and one data element does not dominate all the others. In the data-farming case, the data is split up into many more subelements (for example, region of interest in the picture) than there are workers processors, and in this case, it represents specific processes. The farmer processor transmits the work packets on demand until none are left, and the workers are no longer processing any data. Then, it waits for a result from a worker and immediately sends another work item to it. On the other side, each worker simply receives a work packet, processes it, and returns the result to the farmer until it receives a stop condition from the farmer.

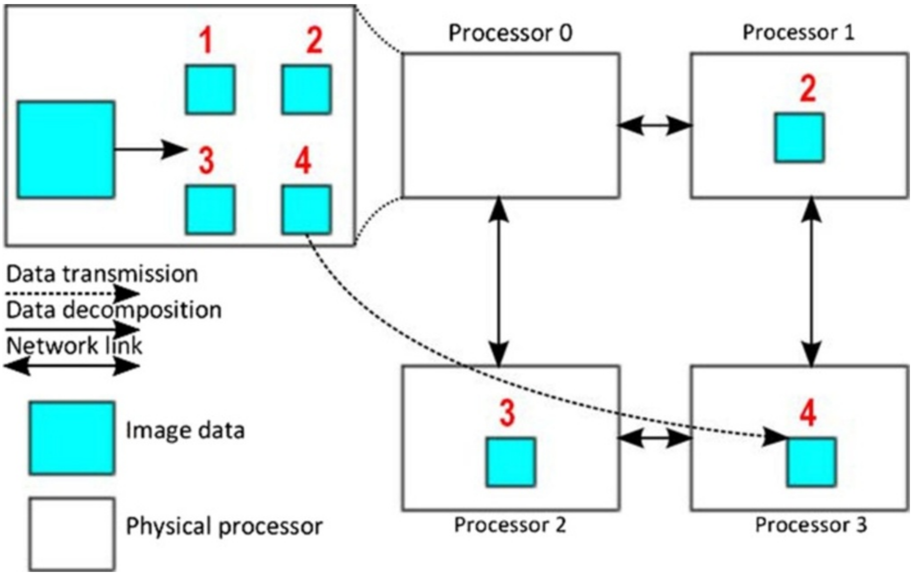


Figure 5 SCM skeleton.

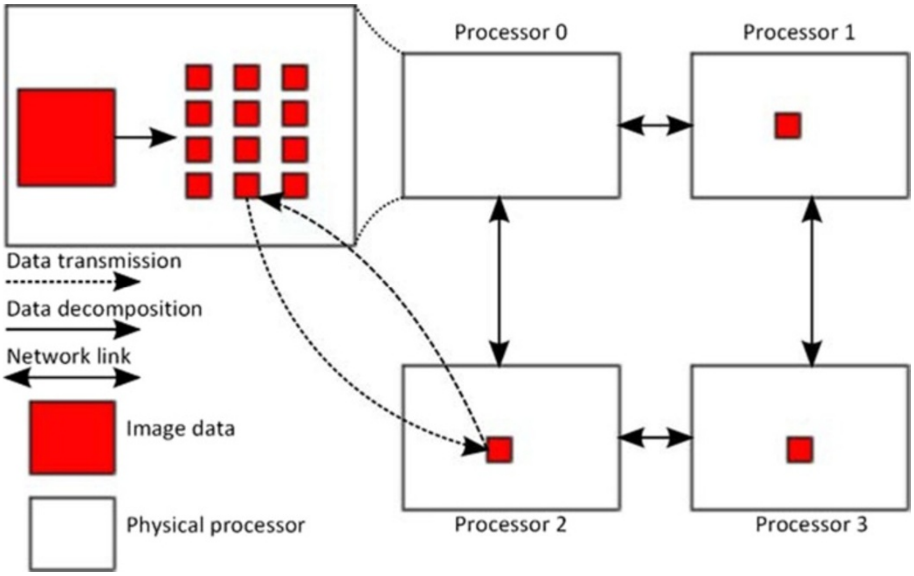


Figure 6 FARM skeleton.

3.4.3 PIPE skeleton

The decomposition of image processing application into many independent and sequential tasks introduces new skeleton (pipe skeleton). Starting from a basic set of skeletons, more complex skeletons can be built by combining the basic ones (SCM, FARM, etc.). Consequently, the various tasks can be computed simultaneously on different pipeline stages. The basic idea of the pipeline skeleton is to split the processing of an application into a series of sequential steps, with storage at the end of each step (Figure 7). This enables the final design to issue instructions at the processing rate of the slowest step, which is much faster than the time needed to perform all steps at once.

In the following, we are motivated to develop a new design flow that enables describing parallel hardware architecture at a much higher abstraction level than traditional hardware description languages.

3.5 Automatic generation of the proposed MPSoC architecture

3.5.1 Design flow overview

The design flow applied for the generation of FPGA-based multiprocessor implementations ideally suited for Xilinx or Altera FPGA platform has seven steps as illustrated in Figure 8 which are the (1) design configurations, (2) design generation, (3) parallelization of sequential application, (4) compilation of the parallel algorithm, (5) design synthesis, (6) design verification, and (7) design implementation.

During the design configurations, the designer has to specify, via a graphical interface 'CubeGen' framework, some architectural parameters including network dimension, MicroBlaze parameterization, memory size allocated to each processor, type of communication link, and use or not of the special IP for I/O (VHDL block designed to control the I/O directly from the video flow). Based on these configuration choices, the CubeGen framework automatically generates the complete system. It includes a homogeneous network of communicating processors (HNCP), the memory components which must be configured to match the application memory requirements (due to program code and image data) of the software application; therefore, communication devices that enable very low latency communication (one using direct point to point links and the other using a hardware router), debug devices which can be configured to test our MPSoC design (UART, JTAG, MDM, etc.) and I/O IPs (from our

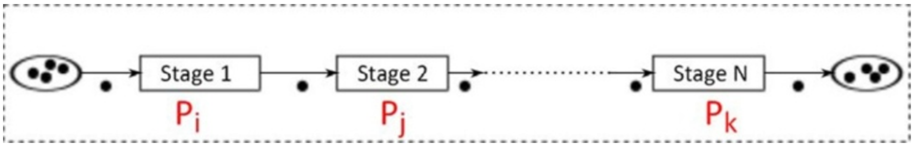


Figure 7 PIPE skeleton.

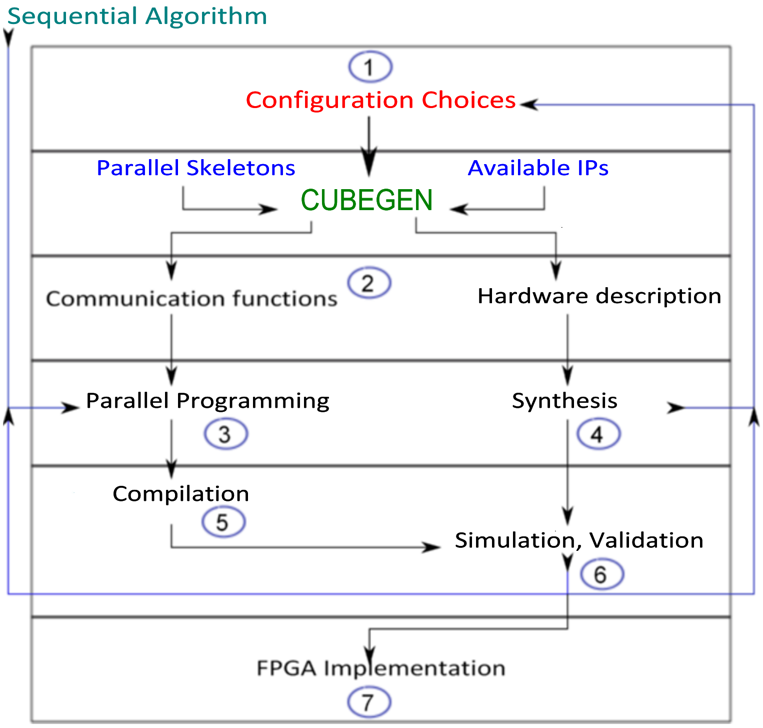


Figure 8 The proposed design flow.

IPs library) which are necessary for processing the video data coming from the PC.

Parallelization of multitask application is based on the main idea that most of the parallel applications were built upon a limited number of recurring schemes of parallelization (parallel skeletons) such as SCM for data or static task parallelism, FARM for dynamic data sharing (or dynamic task sharing), and PIPE for task-parallel parallelism, etc. Taken as input, the application which is programmed using imperative languages such as C, C++, CubeGen framework also generates specific lightweight communication functions that are tuned to the network configuration (number of processors, communication links, parallelization scheme...). Thanks to these communication functions, the designer can easily convert sequential algorithm into a parallel C code. During the compilation, the parallel program is compiled on the HNCP architecture using compiler GCC or G++.

During the design synthesis, the whole system is instantiated on SoPC Xilinx (or Altera) platform using high-level design tools such as the embedded development kIT (EDK) from Xilinx or Quartus II design software from Altera. Afterwards, the HDL code can be verified. Here, if the design does not meet the area constraints, a first loop enables to re-configure and re-generate the HNCP for efficient utilization of resources.

During the design verification, we offer the possibility to simulate the whole system before synthesizing the code. Here, the simulator addresses the evaluation of HNCP architecture performance early in the simulation and validation step. For maximum throughput and performance, the application is executed on the architecture and the execution behavior is captured by using a cycle-accurate industrial RTL simulator (Modelsim). All the relevant characteristics for performance estimation can then be easily extracted: application throughput, latency, utilization of resources, memory utilization, etc.

According to the estimated performance, the designer can change the architecture configuration and/or modify the parallel application to converge to a suitable solution. The designer is responsible for setting these parameters with relevant values according to his background and the technology. The obtained performance results are tightly linked to the chosen parameters to generate the HNCP architecture. Finally, the complete code of the design is implemented in Xilinx Virtex FPGA board (or Altera Stratix FPGA board) using EDK tool (Quartus) to generate the bitstream that configures the FPGA and even upload it.

3.5.2 CubeGen framework for Altera/Xilinx environment

To ease the design of a network of communicating processors, we advance an automated framework to assist the designer in specifying and designing of the homogenous network. The objective is to identify the best configurations choices of multiprocessor on the FPGA for a target application and optimally execute the application tasks with appropriate communication links.

First, the designer starts by specifying their own work environments (Xilinx or Altera environment) and the target platform FPGA (Figure 9a). Second, the designer may select the network dimension and the number of stages of the desired architecture (Figure 9b). Finally, the designer has to specify the different parameters chosen for the network (for each stage), including the soft processor (MicroBlaze, NIOSII) parameterization, memory size allocated to each processor, type of communication link, and use or not of the special IP for I/O (VHDL block designed to control the I/O directly from the video flow) (Figure 10). Actual hardware architecture is done by the choice and parameterization of readily available reconfigurable hardware modules and customizable commercially available IPs. To offer more choices, our library is under development and currently contains for example two soft processors: NIOSII (Altera) and MicroBlaze (Xilinx) and several dedicated HW IPs, etc. Earlier workers aim at including SecretBlaze (soft processor), generic IP Ethernet, etc.

In fact, CubeGen automates the high-level description of the HNCP in few seconds. As result, the developed framework generates the configuration file creation dedicated to the embedded development kit (EDK) of Xilinx Company or to the Quartus II design software of Altera Company. This generated file contains the description of the system with the connection between the different IPs. The randomly selected designs are then simulated to identify the suitable parameter values in order to balance the performance and cost of the desired implementation.

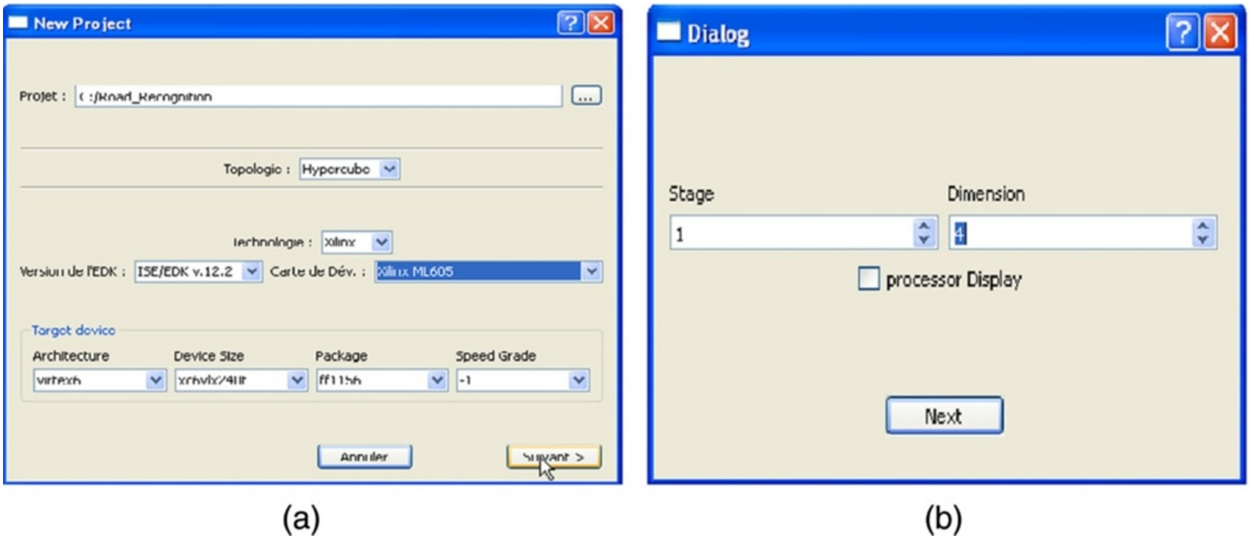
Finally, the designer launches the synthesis of the system with specific target to check if this configuration of the HNCP can be implemented. This methodology matches perfectly with the concept of fast prototyping on SoPC. The designer obtains quickly an architecture tailored for the target application. In addition, CubeGen provides a well-suited library (regarding architecture configuration choice) of specific lightweight communication functions that facilitate conversion from sequential algorithm to a parallel C code.

In the rest of the paper, the effectiveness of the proposed design methodology is shown through parallelized road recognition application on MPSoC architecture.

4 Multihypotheses model-driven approach for road recognition

In this section, we particularly focus on the task of roadside recognition which is widely used for vision-guided vehicle navigation. The proposed algorithm is inspired by the road recognition approach developed by [14]. This application is usually the main part of the lane keeping systems which are designed to provide the roadside location through a video sequence.

Generally, the road edges are characterized by a high-contrast region with low curvature and constant width. With real road images, a determinist description of the road geometry can be a challenge since the road may vary in the form of various road conditions and quality of road markings used to demarcate lane boundaries. Actually, detection strategy proposed by [14] presents a conceptually simple approach for edge detection, followed by edge grouping and pruning. However, as a single hypothesis is used, the recognition process still has problems to follow the lane road through video sequence due to false detections. To keep track of the roadsides, a multihypotheses approach of road recognition can be employed by using



Projet : C:/Road_Recognition

Topologie : Hypercube

Technologie : Xilinx

Version de l'EDK : ISE/EDK v.12.2 Carte de Dév. : Xilinx ML605

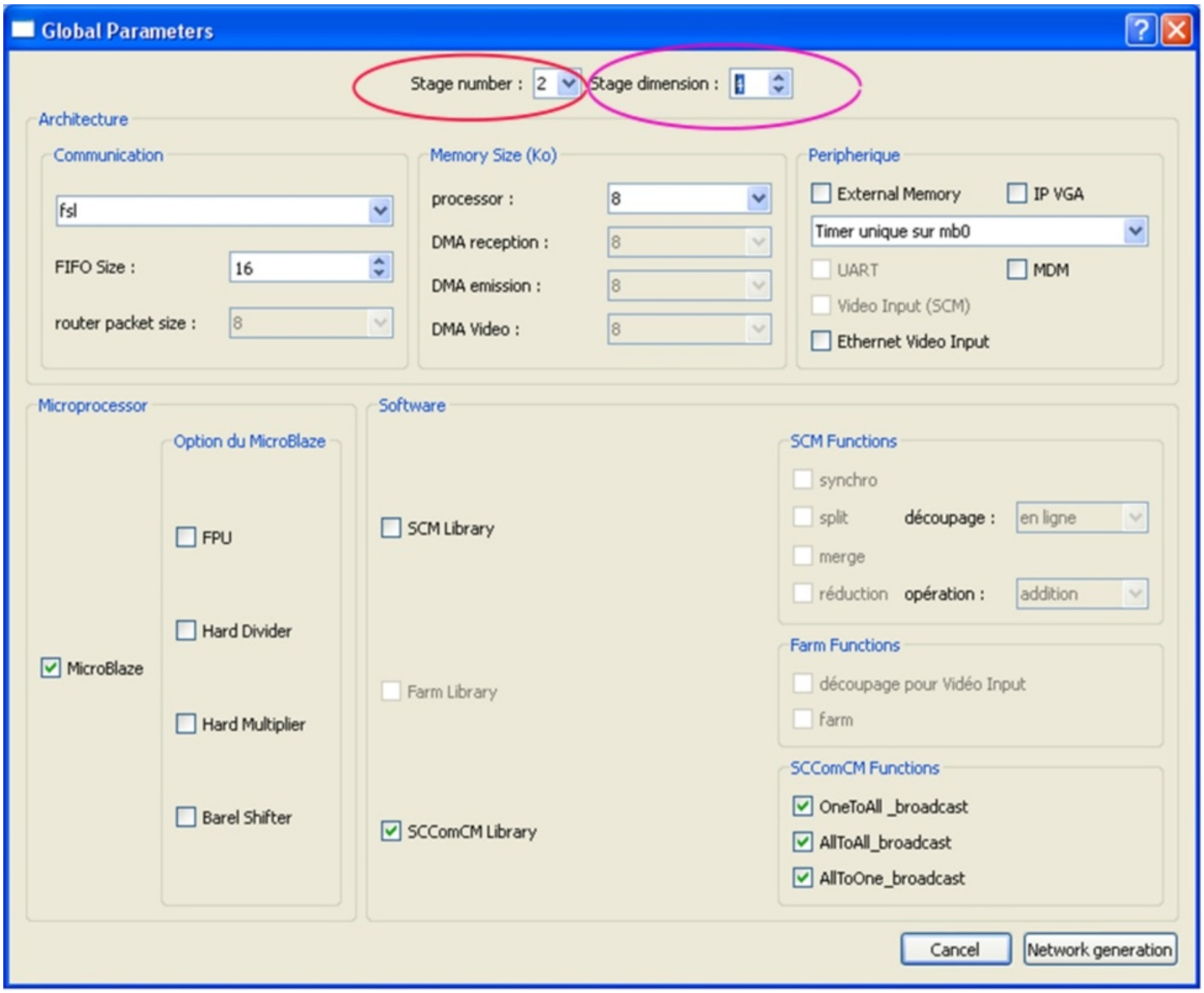
Target device

Architecture	Device Size	Package	Speed Grade
virtex6	xc6vlx240t	ff1156	-1

Stage: 1 Dimension: 4

□ processor Display

Figure 9 GUI of the CubeGen tool. The target platform, the network topology **(a)**, the number of stage and the network dimension **(b)**.



Stage number : 2
Stage dimension : 1

fsl
FIFO Size : 16
router packet size : 8

processor : 8
DMA reception : 8
DMA emission : 8
DMA Video : 8

□ External Memory
□ IP VGA
Timer unique sur mb0
□ UART
□ MDM
□ Video Input (SCM)
□ Ethernet Video Input

☑ MicroBlaze
□ FPU
□ Hard Divider
□ Hard Multiplier
□ Barel Shifter

□ SCM Library
□ Farm Library
☑ SCComCM Library

□ synchro
□ split découpage : en ligne
□ merge
□ réduction opération : addition

□ découpage pour Vidéo Input
□ farm

☑ OneToAll _broadcast
☑ AllToAll_broadcast
☑ AllToOne_broadcast

Figure 10 Architecture parametrization.

various classifications of zones of interest (representative of the road region in image). In fact, this method allows eliminating all the false detections of road edges. The algorithm of sequential recognition is obtained on the basis of the recursive recognition approach presented in [14] applied to accomplishing the different hypotheses in order to identify an optimal hypothesis before the algorithm converges. In addition, the recognition of the roadsides is performed by two main steps: learning and multiple hypothesis recognition. The main feature of this algorithm can be summarized into three phases:

- The road model initialization or learning step is presented for providing a good initial position for the lane model in the image. This algorithm is robust to noises, shadows, and illumination variations in the captured road images and is also applicable to both the marked and the unmarked roads. In practice, the training information will be deduced directly from vehicle localization parameters and camera's parameters.
- The lane recognition problem is formulated by using a set of hypotheses to determine the parameters of road model. Using the detected lines and the image gradient, lane refinement is performed using IIR filter and least median of squares to give more exact lane position by only a local search.
- The road model is updated in time each frame as well as updated from the measurements via a Kalman filter.

4.1 Model of road edges

To build the road model, first, the road region is divided into $2P$ interest regions as shown in Figure 11 and then consider that we have P coordinates v_j for $j \in [1, P]$ to identify the locations of these regions. To simplify, these

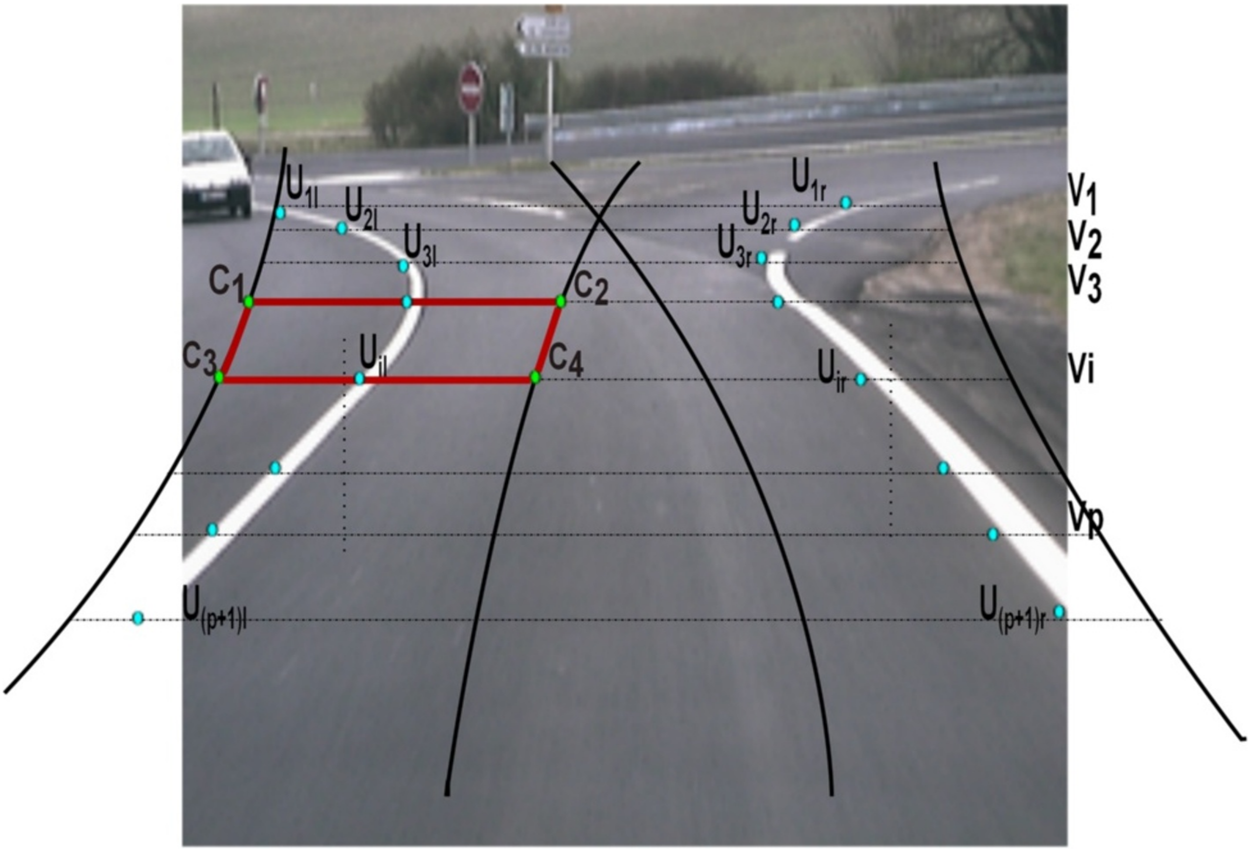


Figure 11 The roadside model. The roadside model deduced from the localization parameters.

coordinates remain fixed in the same locations through a video sequence. The roadside then has an associated model represented by a vector $X_d(P+1)$ and a covariance matrix $C_{X_d}(2P+2)$ which can be represented in the following form:

$$X_d = \left(u_{1l}...u_{(p+1)l}, u_{1r}...u_{(p+1)r}\right),$$

$$C_{X_d} = \begin{pmatrix} \delta^2_{u_{1l}} & . & . & . & . & . \\ . & . & . & . & . & . \\ . & \delta^2_{u_{(p+1)l}} & . & . & . & . \\ . & . & . & \delta^2_{u_{1r}} & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & \delta^2_{u_{(p+1)r}} \end{pmatrix}$$

where u_{il} and u_{ir} ($i \in [1, P+1]$) represent the horizontal image coordinates of the left and right roadside, respectively, for different image rows v_j for $j \in [1, P]$. To reduce the search space, the $\delta^2_{u_{il}}$ and $\delta^2_{u_{ir}}$ define a confidence interval of the possible locations of u_{il} and u_{ir}, respectively, in real road image for $i = \{1...P+1\}$. The initial value of the model (X_d, C_{X_d}) is obtained by assuming all parameter dispersion of a typical road during learning phase.

4.2 Learning phase

The algorithm has an offline learning phase, independent of the road image. In fact, the learning step is used to obtain an initial estimation of the roadside model (X_d, C_{X_d}) according to the vehicle environment (Figure 12) in order to limit the search region of roadsides in the image. This phase is necessary to seek only realistic road configurations in the image. The roadside model is then performed as the relationship between u_i coordinates of roadsides in the image and v_i coordinates, vehicle localization parameters (x_0, ψ), camera inclination parameter (α) and the road geometry (C_l,L):

$$(u_{il}, u_{ir})^t = G\left(v_i, x_0, C_l, \psi, L, \alpha\right)$$

The vector X_d is calculated as the average of the essential parameters $(v_i, x_0, C_l, \psi, L, \alpha)$ established from initial knowledge on the vehicle, the road, the camera, and various ordinates v_i in the image. Variations of the parameters $(x_0, C_l, \psi, L, \alpha)$ according to a normal law in an interval of dispersion wished will led to a set of realistic road configurations in the image. From these various configurations, it is easy to extract the covariance matrix C_{X_d} by using the Jacobian J_d of the function G [14]. The search zone is limited in size once the road model is initialized

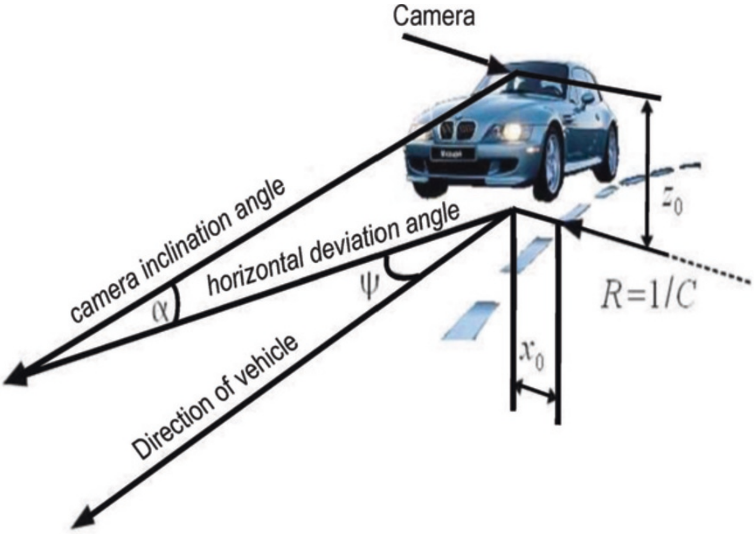


Figure 12 Localization parameters of vehicle in the image.

in the first image during the learning phase; we track the road lanes using the updated road model computed from previous images.

4.3 Recognition phase

For a real image with curved roadsides, the proposed method successfully extracts the current position of the roadsides by improving the detection of road edges. To this end, we generate multiple hypotheses representing possible locations for roadsides. During this phase, we would evaluate all possible detections of the road edges to ensure that we find the correct positions of the roadsides.

To answer to this need, the road area is divided into P interest zones representative of the entire roadsides. The interest zones are then defined by trapezoid forms $(C_1(u_i - \sigma_i, v_i), C_2(u_i + \sigma_i, v_i), C_3(u_{i+1} - \sigma_{i+1}, v_{i+1}), C_4(u_{i+1} + \sigma_{i+1}, v_{i+1}))$ where σ_i is the variance of u_i deduced from C_{X_d} for $i = \{1...P+1\}$ (Figure 11). In our case, we have used 11 interest zones representative of the roadsides. The first stage of the algorithm consists in initializing the roadside model (X_d, C_{X_d}). According to confidence intervals deduced from the covariance matrix C_{X_d}, the different interest zones are ranked according to their size. The zone of greatest interest has the smallest variance of the covariance matrix. Then, our approach relies on the use of multiple hypothesis recognition, corresponding to alternative ways of regrouping the different interest zones. The algorithm flowchart of the recognition iteration using six interest zones is shown in Figure 13. More precisely, a hypothesis refers to one possible way of grouping the zones of interest.

We start by treating the first hypothesis; here, we determine the area of highest interest. If a segment is detected in this zone, the model is then updated to obtain new road model $(X_{dm}, C_{X_{dm}})$; else, the detection is attempted in a new zone defined by the next smallest variance of the covariance matrix and so on. Then, the recognition algorithm is an iterative procedure; a new optimal zone is located according to its variance followed by the detection and the update operations. Finally, if all the interest zones are tested, the algorithm considers that the roadsides are found in the image if a certain criterion is reached.

In a similar way, we can obtain more candidate roadside models for the other hypotheses. The search process (choice of the interest zone, detection, and update) is recursively re-iterated for each hypothesis with different combinations of the interest zones. Depending on this recursive recognition of each hypothesis, we assign them a score. Finally, the recognition scores of all the hypotheses are collected, and the road model with maximum score is selected to be the final estimation for the roadside. As a result, we obtain the optimal value of X_{dop} and $C_{X_{\text{dop}}}$ representing the roadsides in the image. The calculated roadside model becomes the reference model for the first frame, and it is updated in every new input image.

In the following, we will describe the strategies followed to parallelize a specific image processing application (the problem of road recognition) using the proposed generic MPSoC design methodology with refinement of a generic parallel architecture model to meet the specific application computation and communication needs.

5 Parallel implementation of recognition algorithm

This work presents various FPGA implementations of the proposed video processing application which not only achieve the desired latency but also further improve the tracking performance which depend mainly on the

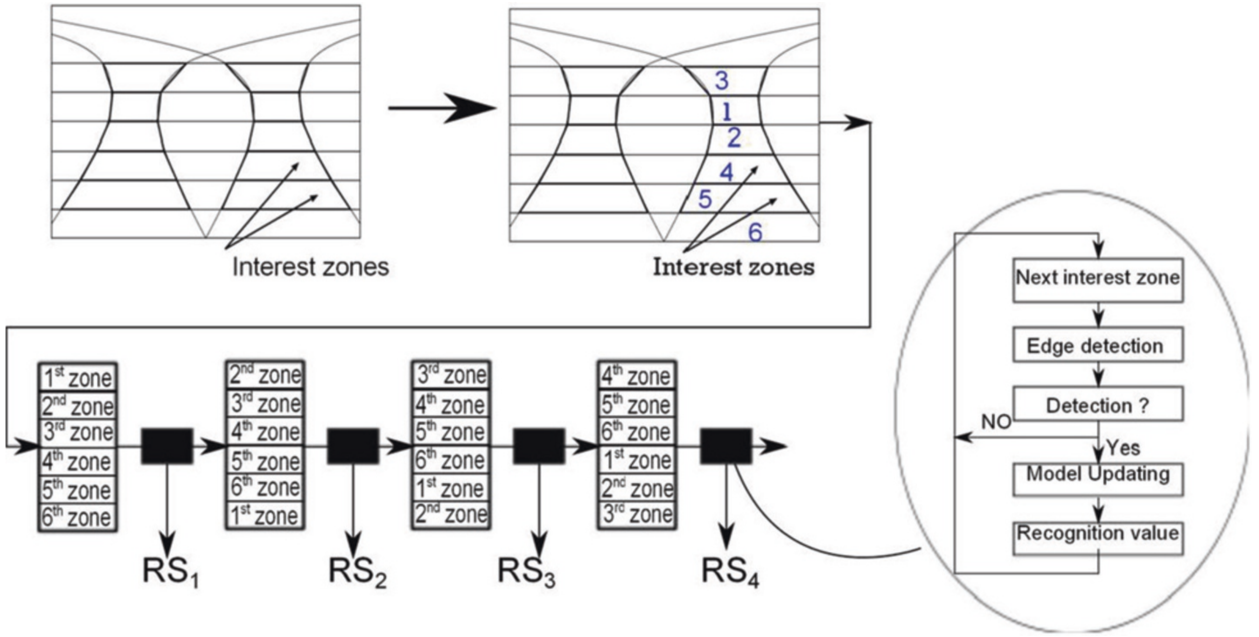


Figure 13 Multihypotheses recognition approach.

number of hypotheses to be executed as well. Within the desired frequency range, the hardware implementation must operate at a maximum frequency of 10 Hz. Additionally, our proposed homogeneous system can be implemented on any FPGA as long as it is large enough. In fact, the memory resources will be the limiting factor in our design with increasing the network dimension especially if the program code requires important memory size. Evidently, if we want to make an application go fast, we must first understand what it spends time doing. Consequently, most of the work is in enhancing and modifying the program code to be executed by the different nodes. We have parallelized a multihypotheses model-driven approach for road recognition application, and embedded it in the presented architecture since the various hypotheses can be done concurrently. As already mentioned, the proposed multihypotheses recognition process tries to coincide with the road boundary with the acquired image content, minimizing the time required for convergence in general and the hypothesis with the best score is the process output.

Depending on the structure of the application, we present the hardware implementations of the road recognition on two separate platforms: (1) a multicore processor based on hardware router as device communication and (2) a multistage architecture based on FSL links as device communication. Then, we will discuss and evaluate the fast prototyping facilities allowed by the proposed approach described in the previous section. Using homogeneous and pipeline computing, it is possible to achieve high computational performance and satisfy the communication needs while the target system remains relatively inexpensive in terms of FPGA occupation, memory size, and power consumption, etc.

5.1 Implementation-based FARM skeleton

The aim of the parallel implementation is to improve the performance by executing the same operation on a set of data elements (zones of interests) transferred to the different N available processors. As shown in Figure 14, the parallel implementation scheme is based mainly on data parallelism (interest zones) between the N available processors. To take advantage of this kind of parallelism, we have to use dynamic data with the enhanced FARM skeleton provided by our skeleton library, which suggests working with a larger number of processing nodes where one processor is selected to be the master and the remaining processors serve as slaves. Starting with this idea, we have to generate network on chip (NoC) based on a packet router, by varying the number of processors. The principal reason why we have chosen hardware router as communication device is its ability to send parts of input image (interest zones) with variable sizes. Therefore, we have to mention here that only the processor 0 can receive the road image. Generally, from a specific network configuration (size of hypercube, communication link, parallelization scheme...), a library of communication functions are automatically generated by CubeGen tool for all skeletons. For FARM skeleton, our library offers a pre-implemented function dedicated to FARM implementation for initialization (Init-FARM), synchronization (Synchro), and work distribution (FARM) as illustrated below.

```
 void main_processor_master_0()
 {
 preprocessing(picture);
 init_farm();//Initialization
 synchro();//the master node sends synchronization pulses to slave nodes to wake up.
 while(Nb_processor_retour < Nb_max_of_Data){
 microblaze_disable_interrupts();
 if(NB_data_slave>0){
 NB_data_slave --;
 microblaze_enable_interrupts();//Block the send or the receive until the slave
 executes the task and delivers
 the result back to the master.
 compute _function_master();
 NB_data_master--;//The master itself process the next data.
 microblaze_disable_interrupts();//Wait for the request to send or to receive.
 farm();
 Nb_processor_retour++;//Receive/send incoming data(outgoing data from/to slave.
 microblaze_disable_interrupts();
  }
  }
  postprocessing(result);
}

void main_other_processors_slave() {
 synchro();
 init_farm();
 farm();
 compute_function_slave();
}
```

The designer can use directly these functions to shorten parallel programming. Thus, we will focus only on implementing our recognition algorithm. At training time, we are presented with *Nb_max_of_Data* interest zones which represents the work item, and in the task-farming case, it represents specific processes. Here, the farmer (processor 0) starts by sending a work item to each worker (from 1 to N). Then, it waits for a result from a worker and immediately sends another work item to it. This repeats until no work items are left, and the workers are no longer processing data. The number of zones to be tested is *Nb_max_of_Data*. On the other side, each worker simply waits for a work item, processes it, and returns the result to the farmer until it receives a stop condition from the farmer. More precisely, the search process (detection and update) is recursively re-iterated by each processor worker for each new interest zone received (Figure 14). Finally, we compare the output of the different search hypotheses looking exhaustively for the best solution. Thereafter, the best estimation of X_d and covariance matrix C_{X_d} (resulting from the recognition step) for the input image will be used in the next frame.

The main feature of this algorithm is that portions of the database (zones of interest, roadside models, and score of recognition) must be communicated between processors to accomplish the necessary treatment. With regard to the latter, the recognition processes are suitably mapped onto a hypercube topology. However, the communication cost of this application has continuously increased, especially when we increase the dimension of homogeneous architecture based on hardware router. Inspite of the interesting gain obtained with this architecture, the road boundary changes with the running of the vehicle across the frame especially when we treat realistic images due to the variation of the roadside region characteristics and vehicle environment. In this situation, the algorithm application has no chance to converge and must be re-initialized (i.e., a naive solution would be to re-run the learning phase when we lose the roadsides). In the following section, we present our parallel multistage architecture which brings solution for the abovementioned problem with the possibility of finding compromises, in terms of communication cost and performance.

5.2 Implementation-based SCM skeleton

For the quality of road edge detection, the geometrical knowledge of the roadsides to follow makes recognition easier. Once the network architecture has been trained using these initial localization data, it is used to recognize roads in all the next frames. Ideally, these parameters remain constant throughout one recognition process (i.e.,

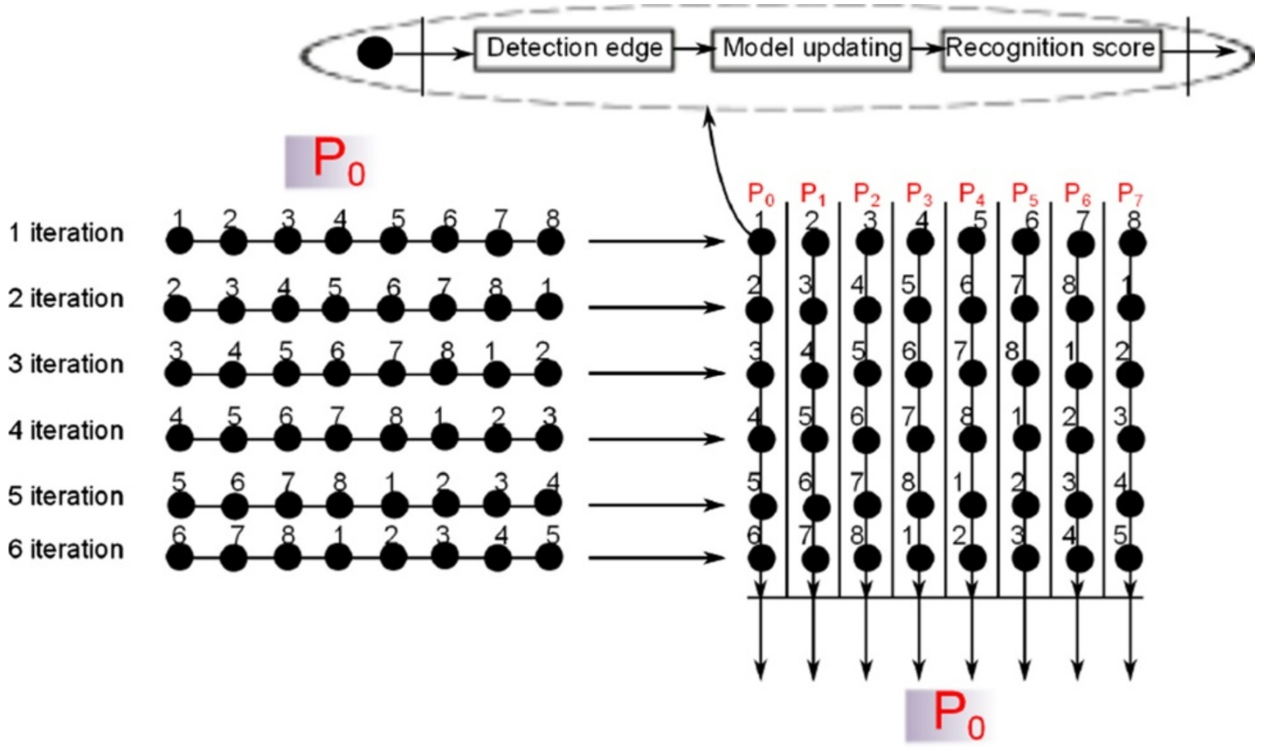


Figure 14 **Data parallel implementation-based FARM skeleton.**

we necessarily keep the same geometric information during the vehicle traveling). Considering a camera fixed on the front of a vehicle, the road boundary changes with the running of the vehicle across the frame dependent in the geometric information available in real scenes. Therefore, the pipelined MPSOC architecture is used to prevent this problem. The idea is interesting because it shows a technique that exploits distributed processing to improve recognition performance.

Since the communication needs for this application are important and computation needs are much heavier, we have chosen the simplest hardware communication solution (using point-to-point connections). Thus, the multistage architecture-based point-to-point links are used since it satisfies the application communication needs for relatively low implementation costs and reduces the communication overhead to have little effect on the global computation time. To this end, the developed video bus drives the input image through the parallel architecture where all processing nodes are independent and perform different image sizes. Thanks to CubeGen framework, the parameterization of the parallel architecture makes it possible to match the specific application needs in terms of BRAM memory and FIFO size that need to buffer large amounts of data. In addition, the two-pipeline architecture represents staged computation where parallelism can be achieved by computing different stages simultaneously on different inputs, mainly when dealing with more than one side simultaneously.

The parallel implementation scheme is based mainly on task parallelism between the N available processors. The different processors supplied by the road image run a single hypothesis to compute one candidate model of roadsides. In this case, our application can be referring to one or more independent tasks running concurrently. In other words, each hypothesis may execute a distinct task. The number of zones to be tested for each task is *Nb_max_of_Data*. Consequently, the parallel execution of multihypothesis process is simply the concurrent execution of independent tasks (or hypotheses) in the different available processors. The above presented idea is shown in Figure 15. With the goal of facilitating the parallel programming task of the proposed pipeline architecture, we use our skeleton library which contains a set of communication functions ‘Split’ and ‘Merge’, ‘Pipe’ functions, and two additional functions called ‘Request’ and ‘Acknowledge’ allowing to execute a ‘synchronization barrier’. Given an application, a set of processing nodes, and a set of communication functions, the most important features of this proposed parallel implementation method can be summarized as follows:

- Broadcast the known roadsides model containing the vector X_d and the covariance matrix C_{X_d} from the

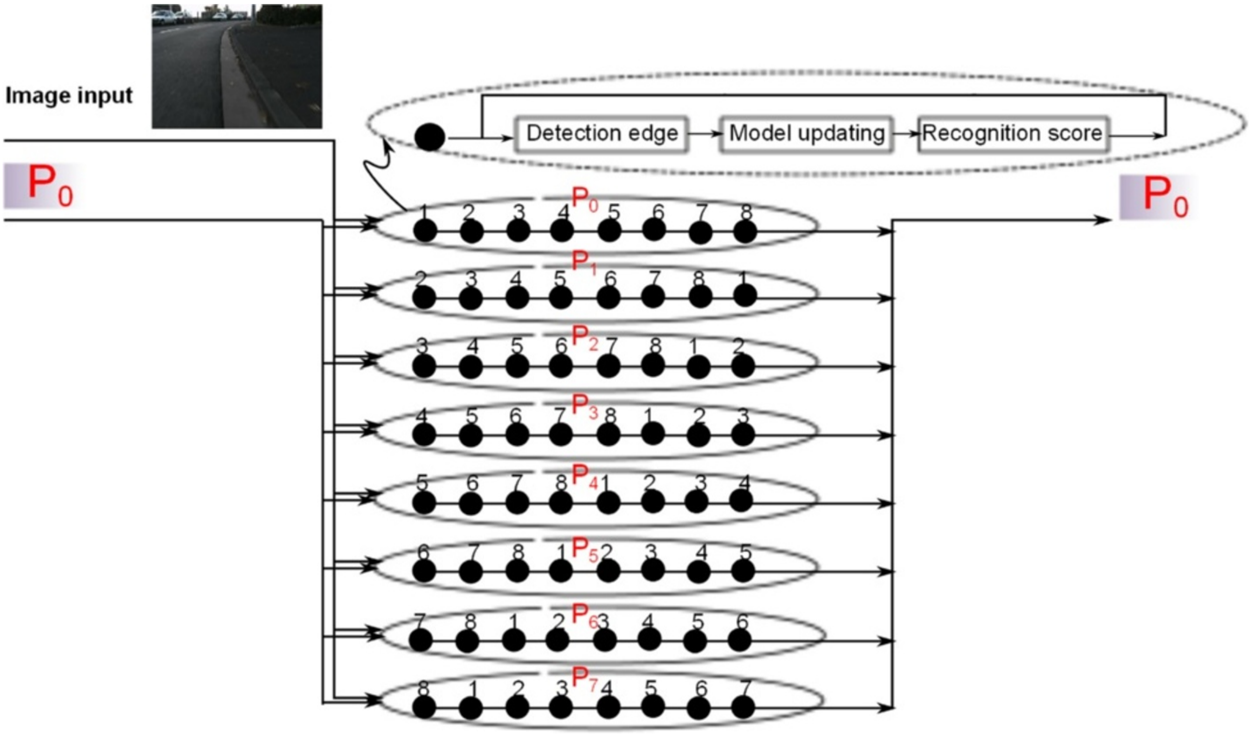


Figure 15 **Task parallel implementation-based SCM skeleton.**

processor 0 to all PEs in time regarding the previous roadside model.

- Apply the recursive recognition process including the two measurement detection and updating in parallel at each PE. In parallel, each PE generates the different primitives (or interest zones) with respect to the received roadside model and sets the recognition value to its locally calculated detection edges.
- The search process on the distributed set of hypotheses in parallel is very quick. Finally, they return their results to the processor 0 which merges them to get the final result.
- The processor 0 compares the roadside models (outputs of the different search hypotheses) to look for the best solution. The roadside model with maximum score of recognition is selected to be the final estimation for the roadside.

In the following, the experiments show that the optimization parameters using our methodology which can increase the performance efficiently in terms of execution times, power, and area. The results show that the methodology allows designers to explore the MPSoC design space more efficiently with the accurate MPSoC profiling information.

6 FPGA implementations

In this section, we use our framework to realize FPGA implementations of the proposed model-driven approach for real-time road recognition mapped onto Xilinx Virtex-6 ML605 FPGA. Additionally, we report some of our results concerning resource costs and the performance of the different parallel homogeneous architectures generated using CubeGen. The main objective of this work is to design an optimal hardware architecture that provides the necessary flexibility and performance to best fit the application needs.

The road recognition algorithm-based model-driven approach has been tested on real road images. In the images, it is possible to see that the algorithm is capable of keeping track of a roadside over a large number of frames (more than 1,200 input images) even in the presence of objects of similar colors and under large variations on the appearance and shape of the roadsides (i.e., road images include curve and straight road, with or without shadows and lane marks).

6.1 Sequential implementation

As explained, the complexity of our application depends mainly on two factors: the number of interest zones and the number of hypotheses. In our experiments under sequential mode, the number of search windows (interest zones) for each road side is set to 8 in order to be better concentrated around the true state.

In addition to the time required for recognition process, we have to consider the communication overhead. With regard to the microprocessor, we use a 32-bit RISC soft core processor, MicroBlaze, which is developed as soft IP by Xilinx Company, running at 200 MHz with 64 k data and instruction memory (BRAM). It can be easily implemented in FPGA-based system. For the sequential implementation on FPGA platform, the proposed hardware architecture has been implemented on a Xilinx Virtex-6 ML605 FPGA. In a video sequence, this implementation has an output latency of approximately 269 ms in the first image by using four hypotheses. Consequently, the complete acquisition and processing of the next input image takes less than 269 ms. The search zone deduced from the C_{X_d} matrix is limited in size for all other frames.

Figure 16 shows the processing flow for road recognition applied to a single image. The model update/detection steps will be repeated until all the interest zones in the right side of the road will be processed. Our results show that it is possible to achieve real-time tracking even operating at relatively low clock frequencies. As mentioned before, one can often expect and frequently achieve an improvement in performance by using far more hypothesis.

6.2 Parallel implementation

Considering the application computational and communication needs, our goals now are to show the impact of taking several configurations choices and therefore to demonstrate the validity of our original MPSoC. In this section, we compare our synthesis results with two different parallel implementations of multiple-hypotheses approach of real-time road recognition on FPGAs. Additionally, in order to evaluate the presented solutions previously seen, we will present in the following subsection some measurement results of the developed hardware approach in term of hardware area and clock speed.

6.2.1 Data parallel implementation

Based on the proposed multiprocessor approach, it is possible to implement various parallel FPGA designs in a single chip. Our main implementation of model-driven approach for real-time road recognition employs a Xilinx SoPC development board and involves many MicroBlaze processors in a distributed memory multiprocessor configuration. Figure 17 summarizes the 1, 4, 8, and 16 processing nodes (PNs) architecture resource utilization (FPGA resource usage). The MPSoC hardware architectures (VHDL code) synthesized on Xilinx Virtex-6 ML605 FPGA. Due to the application memory requirements (program code and image data), each node must contain 64 kb of local memory. To satisfy the communication requirements, each soft processor (MicroBlaze) must have 32 kb for send memory buffers and 8 kb for receive memory buffers. The results show that the router DMA communication device takes a significant part of the resources needed by the whole network of processor. This proportion increases progressively from nearly 66% for a 4-node network to more than 75% for a 32-node network [15]. As expected, increasing the number of core leads to higher speed up and negatively impacts the area occupied. However, the generation of homogeneous system beyond 16 nodes is not possible because of the memory constraints,

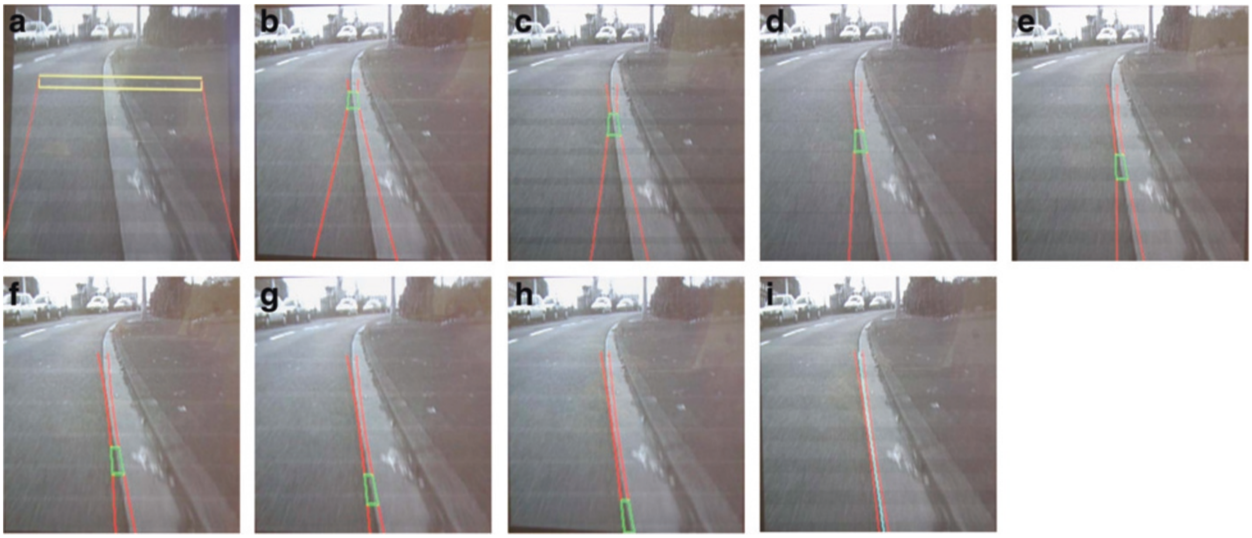


Figure 16 Processing flow for road recognition. (a) Learning phase and the first interest zone, **(b, c, d, e, f, g, h)** the next interest zones, and **(i)** the recognition result.

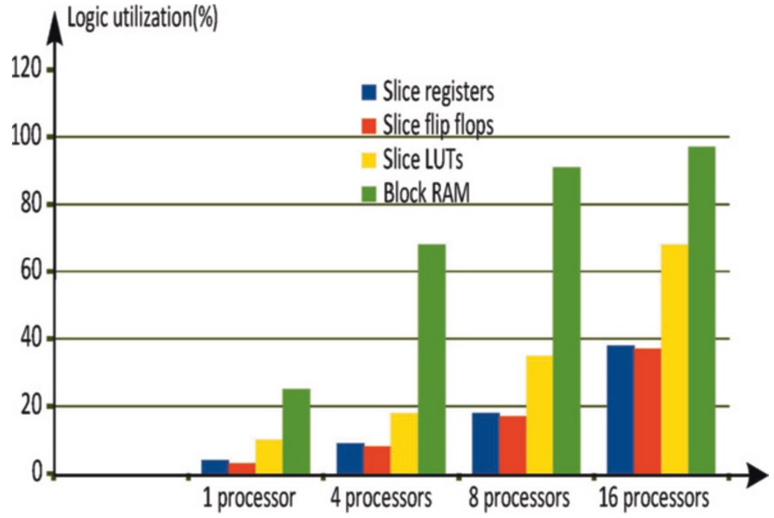


Figure 17 Resource utilization comparison (Xilinx xc6v2x240tff 1156-1).

and FPGAs do not currently contain enough resources to help us implement large parallel designs.

The implementation results for the homogeneous MPSoCs are depicted in Figure 18. The x-axis represents the number of MicroBlaze processors in an MPSoC, and the y-axis depicts the number of clock cycles (in ms). Figure 14 shows the expected performance of our implementation and the experimental execution times on our parallel design with different dimensions (from 1 to 16 processing nodes (PNs)). The time to execute the road lane detection and tracking (recursive process) depends greatly on the number of hypotheses which will be distributed over different computing nodes. Evidently, better tracking performance is delivered by increasing the number of computing nodes, and thus, it allows the user to significantly increase application performance (tracking performance) by implementing more hypotheses. Since the number of available processing nodes is equal to the number of the hypotheses, there is no difference of the theoretical processing times for different architecture sizes (the 4 processor system processes 4 hypotheses in parallel, the 8 processors system processes 8 hypotheses in parallel, and lastly the 16 processors system processes 16 hypotheses in parallel, except that the 1-MicroBlaze system processes 4 hypotheses in sequential). The ideal graph represents a parallel implementation in which all processors operate at 100% efficiency. In reality, efficiency will be less than 100%, and we found some differences in

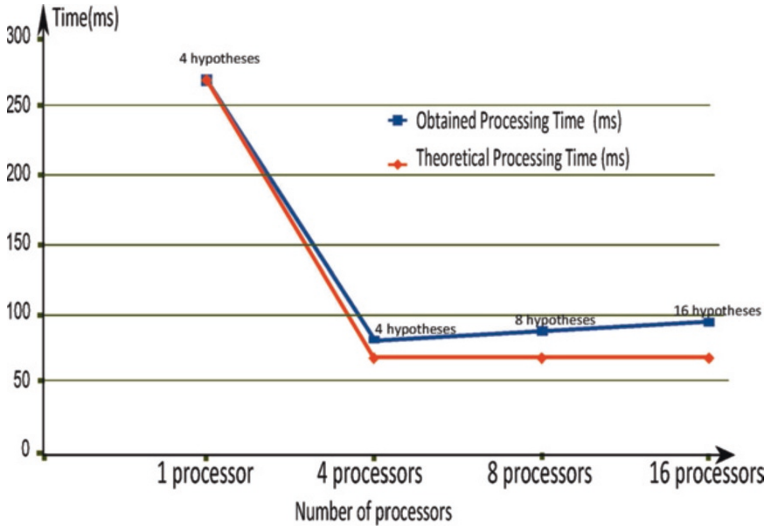


Figure 18 Application execution time (ms) for 1 to 16 processing nodes.

processing time between experimental findings and theoretical assumptions due mainly to communication effects. For data parallelism, a large amount of data is exchanged, which has a larger influence on the final communication time, and must be taken into account. In practice, the latency of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program. For these parallel implementations and from up to 16 processors, communication cost (which is a sequential part) becomes significant. However, the remaining sequential algorithm part in the parallel algorithm (zones of interest generation and evolution step) represents a minor part of the processing time in the sequential implementation, thus allowing an efficient parallel implementation according the Amadahl's law. The improvement in terms of timing and recognition performances of the different generated MPSoCs designs is given by:

$$\begin{aligned}\mathrm{Per} &= (T_{\mathrm{seq}}(1-\mathrm{PN})/T_{\mathrm{par}}(N-\mathrm{PNs}))\\ &\quad\times(\mathrm{Nb.hypotheses.parallel}/\mathrm{Nb.hypotheses.sequential})\end{aligned}$$

where $T_{\mathrm{seq}}(1-\mathrm{PN})$ is the time needed by one single processor to execute the sequential algorithm, $T_{\mathrm{par}}(N-\mathrm{PNs})$ is the time needed by N processors to execute the parallel algorithm, and finally, Nb.hypotheses.parallel and Nb.hypotheses.sequential are the number of hypotheses to be executed on a one single processor system and N processors system, respectively. Based on the experimental results, the performance of 4-MicroBlazes system is approximately equal to 3.44 x. Whereas, the achieved performance by 8-MicroBlazes system is equal to 5.78 x. For 16-MicroBlazes system, our algorithm could achieve a 9.46 x performance improvement compared to a single processor system. As expected, only a limited increase in performance could be achieved by adding more processors due to the communication overhead. In other hand, regarding to the result of experiments results, our proposed design is successful in detecting roadside boundary and correctly tracking target side in all frames despite the changes of vehicle environment. The percentage of correct lane tracking is over 90%, depending on the real road conditions and the number of the hypotheses used. To summarize, the DMA router architecture implementation achieves the best performance and its throughput scales very well as the number of cores increases.

6.2.2 Task parallel implementation

Following the design flow, a number of multiprocessor configurations can be created and programmed easily using our CubeGen framework. In this section, we will evaluate the performance of a novel FPGA implementation of the recognition approach-based task parallelism. The pipeline design described above was simulated and implemented targeting Xilinx FPGA. Furthermore, we present a number of experiments in which we show the FPGA synthesis results and the execution times of the image processing algorithm.

Figure 19 shows the space and resources used by the proposed MPSoC architecture (the pipeline network architecture) in the function of the number of processing nodes in each stage (the cases of 1, 4, and 8 processing nodes (PNs) in each pipeline stage). In these implementations, we have employed the bus video module designed to serve as an I/O engine for MPSOC configurations for varied dimensions. The size of memory for data and program (BRAM) was set to 64 kB. For the four-node architecture in each stage, place and route results lead to an area occupation of 7% for slice registers, 91% for RAM blocks, and 18% for slice LUTs. Since the processors are connected through a hypercube network, each node is associated with frame grabber which contains two buffers (input/output buffer) and supplied with a 64-k local memory. This may explain the results of resource utilization in terms of area occupation. Due to the large size of the contest networks and the limited resources on the Xilinx Virtex-6 ML605 FPGA, we were only able to implement two-pipeline architecture composed of eight PNs in each stage. As expected, the memory resources will be the limiting factor in our design. To improve the tracking performance, the number of hypotheses (tasks) executed in parallel will increase as more processors are added. As a result, the performance is limited by the maximum number of tasks that can be allocated during iteration. However, the communication times to execute the above algorithm onto our hardware architecture should be significantly reduced since the input image is available for all the processors.

Figure 20 provides the execution times of variety of MPSoC architectures, ranging from a one-processor per stage system to an eight-processor system per stage. The ideal graph represents a perfectly parallel program doing the same amount of work in which all processors operate 100% efficiency; communication overheads could be ignored between processors and assumes static task execution times. As expected, the communication overheads have a minor effect in the global processing time, and hence, the performance of the proposed algorithms is very close to the theoretical speedup. The achieved performances are 3.72 x for a four-processor system per stage and 6.58 x for an eight-processor system per stage. Comparing these results to those when using MPSoC design-based hardware router (improvements of 3.72 x and 6.58 x, respectively), this means that the multistage architecture is capable of significantly further improving the parallel implementation of such applications. Thus, the execution time of the recognition application with four computing nodes per stage is approximately 69 ms, whereas it is 269 ms for the original sequential application. The recognition

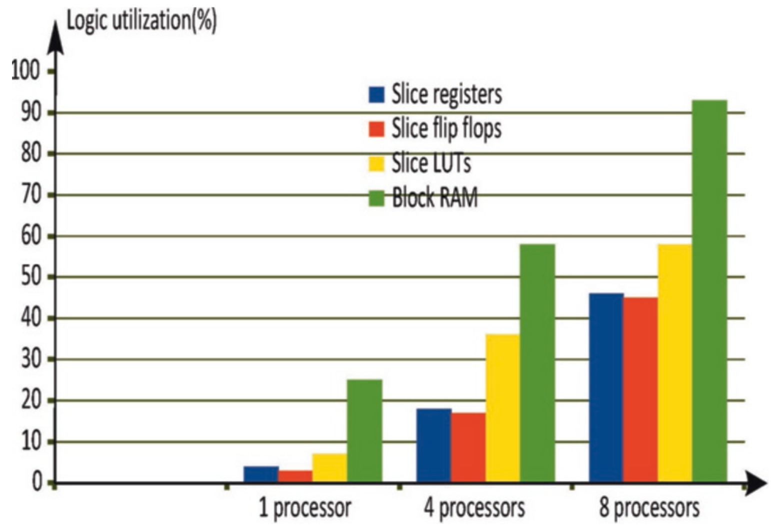


Figure 19 Resource utilization comparison(Xilinx xc6v2x240tff 1156-1).

function requires 53 ms and 16 ms, respectively, for the detection and least median of squares methods in the first frame. Then, their processing times are reduced across all the next frames. Finally, eight-processor system per stage is able to achieve a performance speedup of up to 6.58 x compared to 1-MicroBlaze system. In real practice, we can achieve a speed of at least 18 frames per second. The percentage of correct lane tracking is over 98%, depending on the real road conditions and the number of the hypotheses used.

Finally, although in terms of area both implementations are comparable, our road recognition application is faster than the designs previously presented. Moreover, the presented architecture achieves the best performance, and its throughput scales very well as the number of cores increases. The benefit of this parallel architecture becomes much clearer in the terms of latency and performance.

7 Conclusion

In this paper, we present an improved MPSoC approach to design, test, evaluate, and generate parallel homogeneous architectures that satisfies the severe requirements of real-time image processing applications. For embedding multitasks applications efficiently, we applied our MPSoC-based design flow and design methodology to develop an FPGA-based multiprocessor system with multiple pipeline stages. This homogeneous design aims at balancing the computation requirement and providing enough computing performances to ensure

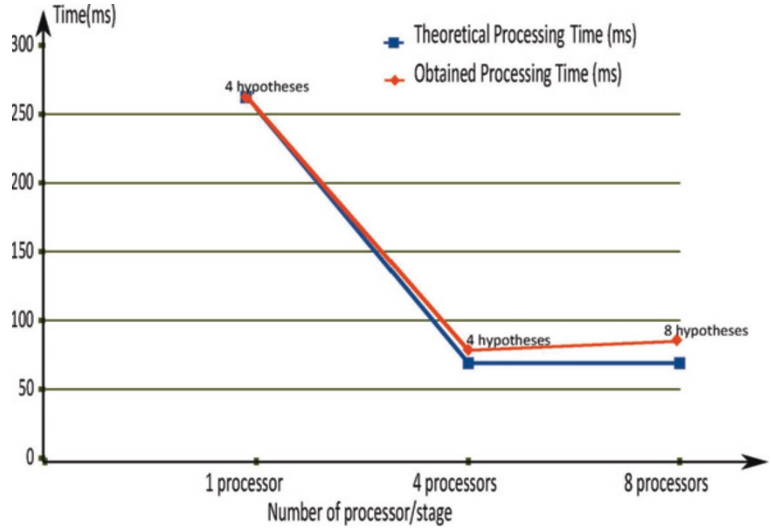


Figure 20 FPGA timing performance results.

real-time processing of complex image/video processing algorithms. Our main objective is to enable quick implementations on the FPGA by developing a new framework that aims at generating an optimal MPSoC for a given application. The user has only to design image and video processing applications in C language and convert them into hardware using our CubeGen tool.

To evaluate the effectiveness of our framework, we have focused on the parallelization and resulting performance of model-driven approach for real-time road recognition on parallel-pipelined architecture based on point-to-point connections. With this work, we have demonstrated that the MPSoC-based system with multiple pipelines achieves a high computational performance and speeds up significantly the execution time using images of a real road scene. The main goal here was to investigate the proposed FPGA tools (both hardware and software tools for rapid development for network architecture) against powerful and complex tasks in autonomous vehicles and robotics.

Similarly, our current work addresses the parallel implementation of multilayer neural network (NNs) applications using our parallel-pipelined architecture. One possible approach to parallelize an application is to pipeline the execution of the sequential hidden layers since our parallel architectures are based on multiple stages whose outputs constitute the inputs of the next stage in processing the whole neural network layer by layer. An important issue in the parallel implementation of artificial neural networks (ANNs) is the communication costs between the different layers. In order to maintain parallel efficiency, we have started by developing the appropriate parallel algorithmic skeletons dedicated to ANNs applications.

On the other side, to extend the choices related to the processing units, we have used an available soft-core processor provided by [16] called SecretBlaze. Consequently, the processor choices will not be limited to the commercial softcores (such as MicroBlaze or NIOSII). This open source 32-bit RISC soft-core processor is based on the instruction set of Xilinx's MicroBlaze. Our motivation here was to explore a new parameterized processor expressed using an HDL (hardware description language) such as Verilog or VHDL allowing for certain aspects of the architecture to be varied.

Competing interests

The author declares that they have no competing interests.

Acknowledgements

This work was funded by the French National Research Agency, the European Commission (Feder funds), Auvergne Region in the framework of the LabEx *IMobS*^{3} and the European Project SEAMOVES.

Author details

[1]Institut Pascal-UMR 6602 CNRS, Blaise Pascal University, 24 Avenue des Landais, Clermont-Ferrand 63177, France. [2]LIMOS-UMR 6158 CNRS, Blaise Pascal University, 24 Avenue des Landais, Clermont-Ferrand 63177, France.

Received: 31 January 2013 Accepted: 30 August 2013
Published: 1 October 2013

References

1. H Chenini, JP Dérutin, T Chateau, Fast prototyping of embedded image processing application on homogenous system - a parallel particle filter tracking method on Homogeneous Network of Communicating Processors(HNCP). VISAPP. **2**, 122–133 (2012)
2. GD Michell, RK Gupta, Hardware/software co-design. IEEE MICRO. **85**, 349–365 (1997)
3. J Vidal, F de Lamotte, G Gogniat, P Soulard, JP Diguet, in *Proceedings of the Conference on Design, Automation and Test in Europe,* Nice, 20–24 Apr 2009. A co-design approach for embedded system modeling and code generation with UML and MARTE (European Design and Automation Association Belgium, 2009), pp. 226–231
4. T Arpinen, E Salminen, TD Hämäläinen, M Hännikäinen, Performance evaluation of UML2-modeled embedded streaming applications with system-level simulation. EURASIP J. Embedded Syst. **2009**, 6:3–6:3 (2009)
5. M Raulet, C Moy, F Urban, JF Nezan, O Déforges, Y Sorel, Rapid prototyping for heterogeneous multicomponent systems: an MPEG-4 stream over an UMTS communication link. EURASIP J. Adv. Signal Process. **2006**, 64369 (2006)
6. O Wander, L Damie, N Gabriela, P Yanick, Y Sungjoo, A Ahmed, G Lovic, DN Mario, Multiprocessor SoC platforms: a component-based design approach. IEEE Design Test Comput. **19**, 52–63 (2002)
7. T Kempf, M Doerper, R Leupers, G Ascheid, H Meyr, T Kogel, B Vanthournout, in *Proceedings of the conference on Design, Automation and Test in Europe, vol.2,* Washington, DC, USA. A modular simulation framework for spatial and temporal task mapping onto multi-processor SoC platforms (IEEE, Piscataway, 2005), pp. 876–881
8. T Kangas, P Kukkala, H Orsila, E Salminen, M Hännikäinen, TD Hämäläinen, J Riihimäki, K Kuusilinna, UML-based multiprocessor SoC design framework. ACM Trans. Embed. Comput. Syst. **5**(2), 281–320 (2006)
9. C Haubelt, J Falk, J Keinert, T Schlichter, M Streubühr, A Deyhle, A Hadert, J Teich, A SystemC-based design methodology for digital signal processing systems. EURASIP J. Embedded Syst. **2007**, 15–15 (2007)
10. N Hristo, T Mark, S Todor, P Andy, P Simon, R Bose, Z Claudiu, E Deprettere, in *45th ACM/IEEE Design Automation Conference,* Anaheim, 8–13 Jun 2008. Daedalus: toward composable multimedia MP-SoC design (IEEE, Piscataway, 2008), pp. 574–579
11. Y Jin, N Satish, K Ravindran, K Keutzer, in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis, CODES+ISSS '05, Jersey City*. An automated exploration framework for FPGA-based soft multiprocessor systems (IEEE, Piscataway, 2005), pp. 273–278
12. K Benkrid, D Crookes, A Benkrid, Towards a general framework for FPGAs based image processing using hardware skeletons. Parallel Comput. **28**(7–8), 1141–1154 (2002)
13. H Chenini, J Dérutin, T Tixier, in *International Joint Conference on Neural Networks 2013 (IJCNN2013),* Dallas, USA. Fast parking control of mobile robot based on multi-layer neural network on homogeneous architecture (IEEE, 2012)
14. A Romuald, C Roland, C Frederic, A model-driven approach for real-time road recognition. Mach. Vis. Appl. **13**(2), 95–107 (2001)
15. L Siéler, L Damez, B Ballet, A Landrault, J Dérutin, in *Proceedings of the 8th FPGAWorld Conference, FPGAWorld '11*. A generic packet router IP for multi-processors network-on-chip (ACM, New York, 2011), pp. 2:1–2:6
16. B Lyonel, VC Luis, B Pascal, T Lionel. The SecretBlaze: a configurable and cost-effective open-source soft-core processor. IPDPS Workshops, Shanghai, 16–20 May 2011 (IEEE, Piscataway, 2011), pp. 310–313

doi:10.1186/1687-6180-2013-153
Cite this article as: Chenini *et al.*: Parallel embedded processor architecture for FPGA-based image processing using parallel software skeletons. *EURASIP Journal on Advances in Signal Processing* 2013 **2013**:153.